PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/60	A1	(11) International Publication Number: WO 98/52132 (43) International Publication Date: 19 November 1998 (19.11.98)

(21) International Application Number: PCT/US98/09009

(22) International Filing Date: 4 May 1998 (04.05.98)

(30) Priority Data:
60/046,173 12 May 1997 (12.05.97) US

(71)(72) Applicant and Inventor: JAMESON, Joel [US/US]; 2607 Greer Road, Palo Alto, CA 94303 (US).

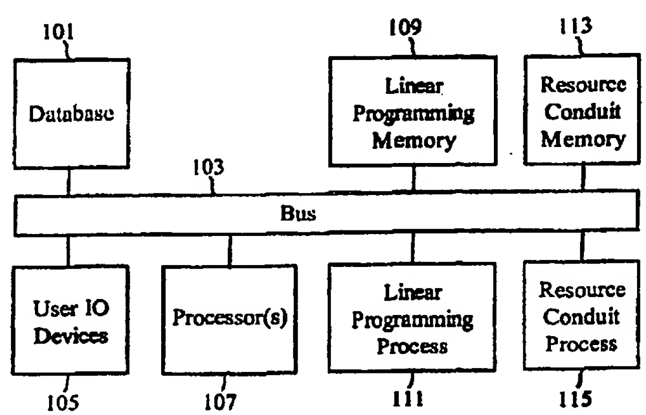
(74) Agents: GERSTEIN, Milton et al.; Hamman & Benn, Suite 3300, 10 S. LaSalle Street, Chicago, IL 60603 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHODS AND APPARATUS FOR ALLOCATING, COSTING, AND PRICING ORGANIZATIONAL RESOURCES



(57) Abstract

This invention is a means both to allocate all types of resources for commercial, governmental, or non–profit organizations and to price such resources. A linear programming process makes fulfillment allocations used to produce product units. A Resource–conduit process governs the linear programming process, uses two–sided shadow prices, and makes aperture allocations to allow Potential–demand to become Realized–demand. A strict opportunity cost perspective is employed, and the cost of buyable resources is deemed to be the opportunity cost of tying up cash. Resource available quantities, product resource requirements, and Potential–demand as a statistical distribution are specified in a database. The invention reads the database, performs optimization, and then writes allocation directives to the database. Also determined and written to the database are resource marginal (incremental) values and product marginal costs. The database can be viewed and edited through the invention's Graphical User Interface. Monte Carlo simulation, along with generation of supply and demand schedules, is included to facilitate analysis, explore "what if", and interact with the user to develop product offering, product pricing, and resource allocation strategies and tactics.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

Methods and Apparatus for Allocating, Costing, and Pricing Organizational Resources

Background Technical Field

This invention relates to methods and systems for allocating resources, specifically to allocating resources in an optimized or near-optimized manner to best serve an organization's goals.

Background Description of Prior Art

As economic theory teaches, every organization — commercial, non-profit, or governmental — has limited resources, i.e., money, raw materials, personnel, real estate, equipment, etc. These limited resources need to be used to best serve an organization's goals. To do otherwise constitutes waste. The business that wastes its resources forgoes profits and risks eventually closing; the non-profit and governmental organizations that waste their resources fail in their missions, fail as institutions, and/or cost their society more than is necessary. As the increasingly competitive world-market develops, and as citizens increasingly question the actions of non-profits and governments, the importance of resource allocation intensifies.

Known methods for allocating organizational resources can be classified as either subjective, accounting, operations research/management science, or miscellany. All of these methods address the same fundamental issue faced by all organizations: which products to make, which services to perform, which projects to undertake, which resources to acquire, and which resources to divest — i.e. all-in-all, which resources to allocate for which purposes. As organizations implement these decisions, physical transformations are made in the physical world. Prices and costs are clearly key factors driving such decisions. As economic theory teaches, given the desires of the populous and the availability of resources, prices and costs are measurements of relative scarcity and serve as a means to direct resources to where they are

best used; this is named "the pricing mechanism" in economics and is a keystone of the free market philosophy.

Under the subjective method, one or more people decide upon allocations in the ways that individuals and groups subjectively decide any matter. This is not objective, nor scientific, and carries with it additional well-known risks and limitations of subjective decision making.

Under accounting methods, so called "costs" are determined and used for deciding issues at hand. As has been well-known for decades, these costs are not economic costs, i.e., the costs that should be used in decision making and that are recognized by economists. By using such invalid costs, undesirable allocations can be made.

The problem with the accountant's cost, as is best known by economists and people with MBAs, is that it:

1. inappropriately includes the price paid for resources, even though such prices are frequently irrelevant to the decision at hand, which is how best to use resources.
2. does not include opportunity cost, which is the loss or waste resulting from not using a resource in its best use.

There is also the famous dilemma of whether cost, as determined by an accountant, should include fixed, sunken, and/or overhead costs. There are strong practical arguments pro and con. Resolution of this issue would significantly affect how organizations calculate costs, and in turn allocate resources. This issue has never been resolved, other than through the dictates of current fashion.

Part of the dilemma of including fixed, sunken, and/or overhead costs is how best to allocate such costs, assuming that such an allocation is going to be made. As is well known, such allocations are largely arbitrary and necessarily distort resulting "costs."

Further, the accounting approach to allocating organizational resources is unable to fine-tune allocation quantities. A priori, it is known that the more of a resource an organization has, the less the resource's marginal (or incremental) value. Accounting offers no means to determine such a marginal value, which is necessary to optimally trade-off resource cost for resource value.

In the 1980s, Activity Based Costing (ABC) was developed to handle some problems resulting from overhead becoming an ever larger component of costs. It is essentially traditional accounting, but with a refined method of allocating overhead costs. It fails to address the above-mentioned problems. ABC is contingent upon all overhead costs being allocated, even though the academic community has for decades argued against such an allocation.

The most important operations research/management science method for allocating organizational resources is linear programming. It was originally formulated by economists in the 1940s and 50s. Part of its promise was both to displace accounting as a method for allocating organizational resources and to resolve the above mentioned accounting problems. For various reasons to be discussed below, linear programming mostly failed to displace accounting as a method for allocating organizational resources. It has largely been confined to use by engineers to solve engineering problems, some of which are organizational allocation problems.

As is well known by practitioners in the field, linear programming is used to allocate some resources for organizations such as oil companies, public utilities, transportation companies, manufacturers, and military units. Though as a method of allocating organizational resources linear programming is very important to some types of organizations for some types of allocations, overall, its use for allocating organizational resources has been limited.

The linearity requirement of linear programming is obviously its most significant deficiency. It cannot handle allocations when economies of scale, economies of scope, or synergistic properties exist; nor can it mix allocating volume and non-volume correlated resources. This means, most importantly, that what are usually known as overhead resources frequently cannot be allocated using linear programming. For example, for a mass market widgets manufacture, linear programming cannot handle the allocation of design resources: a design can be shared by multiple widgets models (economies of scope) and each design used for however many units are sold (economies of scale); further, linear programming cannot: 1) allocate design resources while also considering the effects of design on manufacturing efficiency (synergy), nor 2) simultaneously allocate resources to produce widget units (mix non-volume and volume correlated resources respectively). Practically, this means that linear programming cannot usually be used to allocate some of the most important organizational resources: management time, marketing resources, research and development, product design, product engineering, etc.

Linear programming is not well understood by people likely to make organizational resource allocation decisions. Many of the textbooks published in the mid-1980s contained errors in their explanation of a key concept for using linear programming, even though the concept dates back to the 1950s. See

> Harper, Robert M. Jr.
> "Linear Programming in Managerial Accounting: A 'Misinterpretation of Shadow Prices'"
> *Journal of Accounting Education* 4 (1986) p 123-130.

Some work has been done to facilitate the use of linear programming, but such work has focused on making linear programming easier to use, presuming the user has some general understanding of linear programming. See:

> Gerald Collaud and Jacques Pasquier-Boltuck
> "gLPS: A graphical tool for the definition and manipulation of linear problems"
> *European Journal of Operational Research* 72 (1994) p. 277-286

> Harvey J. Greenberg,

"Syntax-directed report writing in linear programming using ANALYZE"
European Journal of Operational Research 72 (1994) p. 300-311

Asim Roy, Leon Lasdon, and Donald Plane
"End-User optimization with spreadsheet models"
European Journal of Operational Research 39 (1989) p. 131-137

The final problem with using linear programming for allocating organizational resources is that it implicitly assumes a static future in terms of allocations. In other words, once an allocation is made, it is presumed fixed — at least until a new formulation is made and the linear programming process is repeated. This final problem is not addressed by attempts to extend linear programming to handle stochastic or chance-constrained considerations, with or without recourse. Such attempts are focused on making fixed allocations that best endure eventualities. For many organizations, opportunities, available resources, and commitments are in constant flux — there is never a moment when all can be definitively optimized; nor is it generally administratively or technically possible to update formulations and repeat the linear programming process continuously.

What is needed by organizations whose environments are in constant flux is a means to somehow use a single linear programming optimization to make multiple ongoing ad-hoc resource allocations without repeating or resuming the linear programming optimization.

Linear programming has been extended in several overlapping directions: generalized linear programming, parametric analysis/programming, and integer programming. These extensions have concentrated mainly on broadening the theoretical mathematical scope. See:

Karen Aardal and Torbjörn Larsson, "A Benders decomposition based heuristic for the hierarchical production planning problem", *European Journal of Operational Research* 45 (1990) p. 4-14.)

J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems", *Numerische Mathematik* 4 (1962) p. 238-252

George B. Dantzig, *Linear Programming and Extensions* -- Chapter 22: "Programs With Variable Coefficients", Princeton University Press, Princeton (1963)

George B. Dantzig and Philip Wolfe, "Decomposition principle for linear programs", *Operations Research* 8 (1960) p. 101-111

Tomas Gal, *Post-optimal Analysis, Parametric Programming and Related Topics, 2nd ed.*, Walter de Gruyter, Berlin (1995) (Particularly chapters 4 and 7)

Tomas Gal, "RIM multiparametric linear programming", *Management Science* 21 (1975) p. 567-575

Tomas Gal and Josef Nedoma, "Multiparametric Linear Programming", *Management Science* 18 (1972) p. 406-422

Other than integer programming's capability to handle integer variables within a linear programming construct, these extensions are of limited utility and are only for special cases. A general, practical, and useful formulation that utilizes these extensions for allocating organizational resources has not been developed.

Other operations research/management science techniques that might be used as a general means of allocating organizational resources include quadric programming, convex programming, dynamic programming, nonlinear programming, and nondifferentiable optimization. For purposes of allocating organizational resources, these techniques too are of limited utility, and only for special cases. A general, practical, and useful formulation that utilizes these techniques for allocating organizational resources has not been developed.

(For an excellent survey of the techniques of operations research/management science, see:

G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd (ed) *Handbooks in Operations Research and Management Science Volume 1: Optimization* North-Holland Publishing Co., Amsterdam (1989.))

Sometimes standard operations research techniques are adopted, or special techniques developed, to allocate organizational resources. Such techniques and uses are most common in

military, public utility, transportation, and logistic applications. Such methods of allocation are far too specialized to be used outside the areas for which they are specifically developed.

Under the miscellany methods of allocating organizational resources, there is Cobb-Douglas, sequential decision models, the "Theory of Constraints," and internal organizational pricing. The Cobb-Douglas method, used by economists since the 1930s, entails using statistical regression to estimate the following equation:

$$\log q = b_0 + b_1 \log x_1 + b_2 \log x_2 + \ \ldots + \ b_n \log x_n$$

where:

q = quantity of product produced

b_i = estimated coefficient

x_i = resource quantity used

This estimated equation is then used for economic analysis, including determining whether aggregate resource quantities (typically on a national level) should be changed. The problem with this approach is that many data points are required and that it entails a gross aggregation. Furthermore, the formulation, because it completely lacks any linearity, is frequently unrealistic.

For marketing, selling, and advertising purposes, a sequential decision model is sometimes used. A potential buyer is presumed to make a purchase decision in stages, and the goal of the seller is to be able to pass each stage. By depicting the purchase decision as a sequence of stages, such a model helps identify where effort should be focused. Such models are usually qualitative, though they may have probabilities of passage assigned to each stage. Because of its perspective and limited quantification however, the applicability of this method of allocating organizational resources has been limited to only focusing efforts within the marketing, selling, and advertising areas.

"The Theory of Constraints" focuses on identifying a single organizational constraint and then managing that constraint. The problem with this method is its presumption of a single constraint, its qualitative nature, and, to the extent to which it is quantified, its not yielding results or insights any different from applying accounting (variable costing mode) or linear programming.

Sometimes, in some organizations for some resources, an internal price is set by using the above allocation techniques and/or open market prices. (Dorfman, p. 184, mentions using linear programming for such internal pricing.) Such internal prices are then used with the above allocation techniques to determine when and where internally priced resources should be used. When internal prices are set by, and then used in, the above allocation techniques, the techniques' flaws and limitations as previously discussed remain.

All of these methods for allocating organizational resources — subjective, accounting, operations research/management science, and miscellany — are frequently used to allocate resources across several time periods. i.e. used for scheduling. Again, the techniques' flaws and limitations as previously discussed remain. These techniques themselves are frequently deficient, because they are unable to fully optimize resources allocations, given the various dependencies.

In conclusion, these deficiencies have come about because of various unrelated reasons. Traditional accounting reflects the problems, capabilities, and knowledge of the time it was first developed—the first third of this century—prior to most modern theoretic economic understanding. Activity Based Costing limited itself to addressing only some of the most serious problems of traditional accounting. It ignored modern theoretic economic understanding, because practical attempts to use such knowledge were frequently incorrectly done, and, consequently, undesirable allocations were made. Activity Based Costing also ignored modern economic understanding because the economic profession had not sufficiently

bridged the gap between theory and practice. Linear programming never even moderately displaced accounting, because it was not sufficiently theoretically and practically known how to extend and adapt it. Other operations research techniques also never displaced accounting, partly because they were developed to solve special engineering problems and/or as academic exercises.

Hence, today, organizations are without the tools to best allocate resources; and as a consequence, their abilities to reach goals are hindered, the allocation of humanity's resources is sub-optimal, and humanity's living standard is less than what it could be. It is the solution of this problem to which the present invention is directed.

Objects and Advantages

Accordingly, besides the objects and advantages of the present invention described elsewhere herein, several objects and advantages of the invention are to:

1. Optimally, or near optimally, allocate all types of resources belonging to any type of organization to best serve its goals.
2. Provide a means that leads an organization to optimally, or near optimally, allocate all types of resources to best serve its goals.
3. Provide costs, including opportunity costs, that reflect all factors necessary for optimal decisions.
4. Provide resource marginal, or incremental, values that can be used to optimally determine whether additional resources should be acquired or resource levels reduced.
5. Resolve the decades-old dilemma of whether and how to allocate fixed, sunken, and overhead costs.
6. Handle uncertainty when allocating resources and calculating costs and values.

7. Provide an objective means for allocating all types of organizational resources.
8. Adapt and extend linear programming to displace accounting as a means for allocating organizational resources.
9. Unify existing methods of allocating organizational resources.
10. Provide a means to facilitate an analyst in applying economic theory when analyzing organizational resource allocations.
11. Provide a simple means of use that shields the user from complexity.

Still further objects and advantages will become apparent from a consideration of the ensuing description and drawings.

Summary of the Invention

The foundational procedure for achieving these objects and advantages, which will be rigorously defined hereinafter, can be pictured by considering Figures 1 and 2.

Figure 1 illustrates a typical computer configuration: a database 101, a bus 103, one or more user IO devices 105, one or more processors 107, linear programming memory 109, a linear programming process 111 (hereafter, LPP), Resource-conduit memory 113, and a Resource-conduit process 115 (hereafter, RCP). (Figure 1 is explanatory and should not be construed to limit the type of computer system on which the present invention operates.)

Figure 2 shows the Resource-conduit memory 113 in some detail. In this figure, a vector or one-dimensional array *resQuant* contains the available quantities of each resource. A matrix, structure, or two-dimensional array ***rcMat*** contains what are here called groups, such as group 201. Each resource in each element of vector *resQuant* is allocated to the groups in the corresponding column of ***rcMat***. The allocation to each group determines what is here called

an effectiveness, which is typically both between 0.0 and 1.0 and represents a probability. For each row of ***rcMat***, the effectivenesses of each group are multiplied together to determine the elements of vector *rowEffectiveness*. Vector *potentialDemand* contains the maximum conceivable Potential-demand for each of an organization's products; this is what could be sold if the organization had unlimited resources. Each element of vector *rowEffectiveness* times the corresponding element in vector *potentialDemand* determines constraint values (commonly known as original *b* values) fed into linear programming memory 109. Conceptually, these constraint values are termed here Realized-demand.

Once initial linear programming constraint values are determined, the LPP is executed and the following is iterated:

1. the results of the LPP are used to shift or adjust group allocations.
2. new linear programming constraint values are determined.
3. the linear programming memory 109 is updated.

The RCP mainly performs "aperture" allocations, while the LPP mainly performs "fulfillment" allocations. These two types of allocations are defined below. The LPP is a slave of the RCP.

Theory of the Invention

Part of the underlying theory of the present invention is that all organizational allocations can be divided into either fulfillment or aperture allocations. Fulfillment allocations use resources to directly make individual product units. Using resources in this way is commonly deemed to generate so-called direct or variable costs that vary with production volume. Aperture allocations are made to keep an organization viable and able to offer its products. These types

of allocations are commonly deemed to generate so called indirect, overhead, or fixed costs that do not vary by production volume. Conceivably, a resource can be used for both fulfillment and aperture purposes.

The term "aperture" reflects how the present invention deems certain allocations: as allowing Potential-demand to manifest and become Realized-demand. The fundamental purpose of a group is to transform an allocation into an effectiveness. The higher the allocation to a group, the higher the effectiveness, which results in a higher percentage of Potential-demand becoming Realized-demand. For instance, the allocation to:

- Group 201 might be research and development people months; effectiveness is the percentage of Potential-demand that finds the resulting product functionality desirable.
- Group 203 might be product-design months; effectiveness is the percentage of Potential-demand that finds the resulting product design desirable.
- Group 205 might be advertising dollars; effectiveness is awareness bought by such dollars.

For a unit of Potential-demand to become Realized-demand, it must find the functionality desirable, it must find the design desirable, and it must be aware of the product — it must survive a series of probabilities. This sequential process is modeled here by multiplying the effectivenesses of the groups to obtain *rowEffectiveness*, which is in turn multiplied by *potentialDemand* to obtain the constraint value (Realized-demand) used in the LPP.

A group can span several rows of ***rcMat***, and thus the group's effectiveness used for determining the values of several elements of *rowEffectiveness*. This row spanning means that a single aperture allocation can apply to several products simultaneously. For instance, the application of design resources might apply to, and benefit, several products simultaneously.

The rows spanned by groups in one column of ***rcMat*** can be independent of the rows spanned by groups in another column. This independence of row spanning means that products can

share and not share resources in arbitrary patterns. For instance, product groupings to share and not share design resources can be independent of the product groupings to share and not share advertising resources.

The relationship between group allocation and effectiveness can be empirically determined by experience, judgment, statistical analysis, or using a coefficient of a Cobb-Douglas function. Because of the independence of the groups, the relationship between group allocation and effectiveness can be determined independently for each group. The relationship can be determined by answering the following question: "Presuming that a group's allocation is the only factor determining whether a product will be purchased, and made available for purchase, how does the probability of purchase vary as the allocation varies?" Presenting this question and being able to work the answer is a major advantage of the present invention. Heretofore, it has usually been very difficult, if not impossible, to individually and collectively analytically consider and evaluate what are here termed "aperture allocations."

(Management time is one of the most important resources an organization has. Groups can also handle such a resource: the allocation of such a resource to a group yields, as before, an effectiveness, which is the percentage of Potential-demand that survives to become Realized-demand, given that management time has been used to make the product available and desirable.)

Each resource is considered either fixed or buyable. A fixed resource is one that is available on-hand and the on-hand quantity cannot be changed. A buyable resource is one that is purchased prior to use; its availability is infinite, given a willingness and ability to pay a purchase price. A fixed resource named Working Inventory Cash (WI-cash) (loosely, working capital) is used to finance the purchase of such buyable resources. It is the lost opportunity of tying up of that cash that is the real cost of buyable resources — and not the purchase price per se.

For example, owned office space is typically a fixed resource: an organization is not apt to continuously buy and sell office space as "needs" vary. Public utility services are buyable resources, since they are frequently, if not continuously, purchased. Employees can be considered either fixed or buyable resources. If an organization generally wants to retain its employees through ups and downs, then employees are fixed resources. If an organization wants employees strictly on a day-to-day as-needed basis, then they are buyable resources. Note that for all fixed resources, including employees, periodic payments, such as salaries, are not directly considered by the present invention: the invention optimally allocates fixed resources presuming their availability is fixed; current payments for such resources is irrelevant to the decision of optimal allocation. Whether the quantities of fixed resources are increased or decreased is decided exogenously of the invention by the user. To help the user, the invention generates marginal values and demand curves that help anticipate the effects of changing fixed resource quantities.

Though this description is written using a terminology suitable for a commercial manufacturing concern, the present invention is just as applicable for commercial service, non-profit, and government entities. From the invention's perspective, a commercial service is tantamount to a commercial product — both require resources to fulfill a sale. Products and services provided by non-profits and governments also require resources, but are handled slightly differently: because such an organization doesn't usually receive a full price (value) for its products and services, the "price" used in the allocation process needs to include an estimated value to society of providing a unit of the service or product.

As will be explained, the present invention can make allocations to either maximize internal producer's surplus (IPS) or maximize cash. The first term derives from the economist's term "producer's surplus." It's called internal here because the economist's "producer's surplus" is technically a societal surplus. A strict opportunity cost perspective is employed here — IPS is

profit as compared with a zero profit of doing nothing. For non-profits and government entities, IPS is a measurement of fulfilling their missions. IPS both includes non-monetary benefits received by the organization when its products are purchased and includes wear-and-tear market depreciation on equipment. When an organization's survival is at stake, non-monetary benefits and wear-and-tear market depreciation on equipment becomes irrelevant: the only thing that is relevant is increasing cash. For such situations, maximizing cash is the appropriate allocation objective.

As will be explained, the present invention can make allocations either directly or indirectly. In the direct method, the invention explicitly allocates resources. In the indirect method, the invention uses Monte Carlo simulation to estimate the opportunity cost, or value, of each resource. This opportunity cost is then used to price each resource, which determines when and where it should be used.

The major advantage of the present invention is to, for the first time, optimally allocate all types of organizational resources for all types of organizations.

Drawing Figures

In the drawings, closely related Figures have the same number but different alphabetic suffixes:

Figure 1 illustrates an explanatory computer configuration.

Figure 2 shows a conceptual memory layout.

Figure 3 shows a basic database schema.

Figure 4 shows prior-art linear-programming memory.

Figure 5 shows Resource-conduit memory.

Figure 6 shows group head and group element data fields.

Figure 7 shows the basic allocation process.

Figure 8A shows a graphical depiction of allocation movements; Figure 8B shows corresponding allocation shifts in matrix ***rcMat***.

Figure 9 shows the basic initialization process.

Figure 10 shows the Axis-walk process.

Figures 11A and 11B show the Axis-walk allocation shift in detail.

Figure 12 shows the Top-walk process.

Figures 13A, 13B, and 13C show the Top-walk allocation shift in detail.

Figure 14 shows the Lateral-walk process.

Figures 15A and 15B show the Ridge-walk process.

Figures 16A and 16B show the Ridge-walk allocation shift in detail.

Figure 17 shows the basic finalization process.

Figure 18 shows the top portion of the Graphical User Interface (GUI) distribution window.

Figures 19A and 19B show the top portion of the GUI resources window.

Figures 20A and 20B show the top portion of the GUI products window.

Figure 21 shows the GUI results window.

Figure 22 shows the preferred allocation process.

Figure 23 shows the supply schedule generation process.

Figure 24 shows the demand schedule generation process.

Detailed Description — Basic Embodiment

The basic embodiment of the present invention will be discussed first. Afterwards, the preferred embodiment, with its extensions of the basic embodiment, will be presented.

With one exception, all costs mentioned in the present invention refer to opportunity costs, which are derived from the in-progress or finalized allocations. The one exception is expenditures for buyable resources that are written to the database and shown in the GUI windows. Here, the words "cost" and "value" are almost synonymous: cost will tend to be used when a subtraction orientation is appropriate and value will tend to be used when an addition orientation is appropriate. The economist's word "marginal" means incremental or first functional derivative. Pseudo-code syntax is loosely based on 'C', C++, SQL and includes expository text. Vectors and arrays start at element 0. Indentation is used to indicate a body of code or a line continuation. Pseudo-code text overrules what is shown in the figures. Floating-point comparisons are presumed done with a tolerance that is not explicit in the figures or pseudo-code. The expression "organizational resources" refers to resources that are directly or indirectly controlled, or are obtainable, by an organization and that can be used to serve its goals.

Database

The basic embodiment of Database 101 is shown in Figure 3. A simple quasi-relational schema is used here to facilitate understanding. It should be understood that the present invention can easily work with other schemata and database technologies, whether relational or not. There are five tables: Resource, Group, Group Association, Product, and UnitReq. The Resource Table has *nRes* rows and describes available resources: name (*resourceName*), available quantity (*availQuant*), used quantity (*meanUse*) and marginal, or incremental, value (*marginalValue*). The Group Table describes groups: name (*groupName*), resource (*resourceName*), the allocation-to-effectiveness function (structure *atoeFnPt*), allocation (*meanAlloc*), and marginal value. The allocation-to-effectiveness function is described using *nir+1* points, which determine *nir* continuous line segments. These points have only non-negative coordinates and are ordered such that *atoeFnPt[i].allocation* <

atoeFnPt[i+1].allocation, where *0<=i* and *i<nir-1*. (To facilitate exposition, the allocation-to-effectiveness function is presumed to pass through the origin, where *atoeFnPt[0]* is the origin point. Also to facilitate exposition, each group is presumed to have the same number (*nir*) of line segments. Relaxation of these two presumptions requires several small obvious changes throughout the exposition.) The Product Table has *mProd* rows and describes products: name (*productName*), price, Potential-demand, quantity-to-produce as the result of the optimized allocation process (*meanSupply*), and marginal cost. The UnitReq Table describes the fulfillment quantities of resources needed to produce each product unit. The Group Association Table maps a many-to-many association relationship between the Group and the Product Tables.

Memory

Figure 4 shows prior-art linear programming memory 109 in some detail, using standard notation: initially the m by mn matrix $\boldsymbol{a}$ contains constraint coefficients; vector b contains constraint bounds; vector c contains object coefficients; and scalar d contains the value of optimization. (The absolute value of d, i.e., $|d|$, is utilized here to avoid awkward wording.) Within matrix $\boldsymbol{a}$ is the standard rectangular matrix $\boldsymbol{B}$, which, initially is an identity matrix. In the right-hand portion of matrix $\boldsymbol{a}$ are n (*mProd*) columns, each initially containing product resource-requirement coefficients.

Resource-conduit memory 113 is shown in further detail in Figure 5. Matrix ***rcMat*** has m rows and *nRes* columns. The number of products (*mProd*) plus the number of resources (*nRes*) equals m. The vectors *bHold*, *bOrg*, *rowEffectiveness*, and *potentialDemand* each have m elements. Vector *bHold* holds temporary copies of vector b. The vector *bOrg* contains the current linear programming problem's original b vector values — the product of each element in vectors *rowEffectiveness* and *potentialDemand*. The vectors *resQuant*, *rwpDest*, *rwpSour*, *rwOldAlloc*, *rwOldMC* and *dpTieSubBlk* each have *nRes* elements, and each element of these vectors applies only to the corresponding column in matrix ***rcMat***. As explained previously,

vector *resQuant* contains the available resource quantities. The Ridge-walk process, to be described later, entails simultaneously shifting allocations from several groups to several groups. Conceptually, the source and destination groups are in separate rows of ***rcMat***. The vector *rwpDest* contains pointers to the destination groups; *rwpSour* contains pointers to source groups; and vectors *rwOldAlloc* and *rwOldMC* contain pre-allocation-shifting destination allocations and source marginal costs respectively. For each of the *mProd* products, matrix ***dpTie*** has a row containing indexes of Direct-put groups, which are defined below. Vector *dpTieSubBlk* contains boolean values indicating whether the Direct-put groups referenced in matrix ***dpTie*** should not be used in vector *rwpSour*.

The Top-walk process, also to be described later, entails simultaneously transferring resources from several groups to several groups. These groups constitute a chain. The vectors *twpGroupSub* and *twpGroupAdd* identify this chain by containing pointers to groups for which the allocation is decreasing and increasing respectively. The variable *twnLink* contains the number of links in the chain.

The Ridge-walk process uses *rwiRow* as an iterator. Both Axis-walk and Top-walk avoid allocation shifts that result in *rowEffectiveness[rwiRow]* changing. The vector *sumWICash*, with *mProd* elements, contains the required expenditures for buyable resources to produce one unit of each of the *mProd* products.

A group consists of one or more of what are here termed group elements. For each group, one element is a group head, that, besides containing element data, contains data applicable to the entire group. Each row-column position of matrix ***rcMat*** is empty or contains either a group head or a group element. For any group, all elements, including the head, are in the same column of ***rcMat***. There is at least one group head in each column of ***rcMat***. Rows *mProd* through *m-1* each contain a single group head; these groups have only a single element and they

are termed Direct-put groups. Here, groups will be named and referenced by their locations in ***rcMat***.

Figure 6 shows the data contained in group heads and elements. A group head contains all the data fields of a group element; references to elements of a group implicitly include the group's head. A group head contains an allocation and a variable to hold working-temporary allocation values (*allocationHold*). As in the Group Table in Database 101, a group head contains an *atoeFnPt* structure that defines the allocation-to-effectiveness function with $nir+1$ points that determine *nir* continuous line segments. These points have only non-negative coordinates and are ordered such that *atoeFnPt[i].allocation* < *atoeFnPt[i+1].allocation*, where $0<=i$ and $i< nir-1$. Variables *dedaSub* and *dedaAdd* contain directional derivatives of the allocation-to-effectiveness function. Structure *atoeFnPt* is indexed by *ir*. Variables *maxSub* and *maxAdd*, respectively, contain the maximum decrement and increment to the allocation that can be made, such that the directional derivative of the allocation-to-effectiveness function remains the same. Variable *gmcSub* (group marginal cost subtract) contains the marginal cost of decreasing the group's allocation; *gmvAdd* (group marginal value add) contains the marginal value of increasing the group's allocation. Variable *twmcSub* (Top-walk marginal-cost subtract) contains the marginal cost of decreasing the group's allocation, while simultaneously: 1) making a compensatory allocation increase to the group with a head at row *twcRow* and column *twcCol* in ***rcMat***, and 2) making a compensatory allocation decrease to the group with a head at row *twcsRow* in column *twcCol*. The variable effectiveness is the result of applying the allocation-to-effectiveness function using the current allocation; its value is copied to each group element. The variable *effectivenessHold* holds working-temporary effectiveness values. The variable *emcSub*, which is found in both group heads and elements, is the single-row marginal cost of decreasing the group's allocation; the sum of *emcSub* for each element in a group equals the group's *gmcSub*. Similarly, *emvAdd* is the single-row marginal value of increasing the group's allocation. The variable *subBlk*, found in both group heads and elements, is a boolean value indicating whether a reduction in the group's allocation should be

blocked (i.e. prevented) by setting *emcSub* to a very large value. A group head is also a group element.

Basic Embodiment Processing Steps

The basic embodiment processing steps are shown in Figure 7. The initialization process 701 entails loading Database 101 data into both linear programming memory 109 and Resource-conduit memory 113 and doing initial allocations. Process 703 entails executing the LPP. Axis-walk process 705 entails iteratively shifting part of an allocation from one group to another within each column of ***rcMat***. Top-walk process 707 entails shifting part of an allocation from one group to another, while simultaneously making a chain of compensatory allocation shifts. Lateral-walk process 709 entails performing modified Top-walk, and in turn possibly Axis-walk, iterations. Ridge-walk process 711 entails attempting to move from a local to a better, if not global, optimum. The finalization process 713 posts the results to Database 101.

Graphical Depiction

Graphical depictions of the Axis-walk, Top-walk, Lateral-walk, and Ridge-walk processes are shown in Figure 8A. This figure shows the optimization surface holding everything constant, except: 1) the allocations to two single-element groups in the same row k of ***rcMat*** (where $0<=k$ and $k<mProd$) and 2) $c[k]$, which is either, depending on the surface point, 0 or a constant negative value. (Note that this constancy is being pretended. In actual operation, the surface represented in Figure 8A frequently changes as movements take place.) The horizontal axis is the allocation of one resource to one group; the backward axis is the allocation of the other resource to the other group; the vertical axis is $|d|$, the value being optimized. The value of $|d|$ increases as long as either or both allocations increase, up to a saturation level, which once reached, results in no further increase in $|d|$. Such a saturation level is depicted by a contour

curve 801, which passes through a point 835. Figure 8B shows the upper left-hand portion of an example ***rcMat*** matrix, where each matrix element contains a group head. (Figures 8A and 8B and associated descriptions are used here to facilitate understanding, and should not be construed to define or bound the present invention.)

Axis-walk process 705 entails increasing the allocation of one group, as shown in the Figure by moving from a point 803 to a point 805, while decreasing the allocation of another group, which would be similar to moving on that row's surface from a point 807 to a point 809. Such a movement is done until a directional derivative changes. In terms of ***rcMat***, such a movement corresponds to shifting an allocation from one group to another group within the same column, e.g., shifting some of the allocation of Group 821 to Group 817.

In addition to moving parallel to an axis as in Axis-walk, Top-walk process 707 also entails moving along a contour curve such as contour curve 801. Such a movement has one group's allocation increasing, while another group's allocation decreases, such that the mathematical product of the two group's effectivenesses remains constant. With the mathematical product being constant, from the perspective shown in Figure 8A, $|d|$ also remains constant. In terms of ***rcMat***, this might entail, for example, shifting the allocation from Groups 821 to 817, 819 to 823, and 825 to 815. The allocation increase in Group 817 and the decrease in Group 819 leaves the product of the two groups' effectivenesses constant and corresponds to movement along contour 801. (The same is also true for the 823 and 825 group pair.) The decrease in $|d|$, because of the decrease in the allocation of Group 821, is more than offset by the increase in $|d|$, resulting from the increase allocation in Group 815.

Each Axis-walk and Top-walk shift (movement) is done until a directional derivative changes. Such a change occurs when the end-point of an allocation-to-effectiveness line segment, or the edge of a linear programming facet, is reached. The size of each shift is determined by whittling-down an entertained shifting quantity. (The word "shift" refers to shifting an

allocation from one group to another group in matrix ***rcMat***; the word "movement" refers to moving on the geometric surface. Any shift can be pictured as a movement; any movement pictured as a shift.)

Lateral-walk process 709 determines a surface just below the surface depicted in Figure 8A, and then applies and evaluates Top-walk, and indirectly Axis-walk, iterations. This stratagem is needed because the directional derivatives used individually by both Top-walk and Axis-walk may be inter-dependent and result in an instantaneous quantum change upon starting a shift or movement.

The Ridge-walk process 711 entails serially considering each of the *mProd* products, and transferring, at minimum cost, allocations to groups of the considered product (***rcMat row***) in order to force an increase in the product's *rowEffectiveness*. This is done to explore the possibility of moving from one local to a higher, if not global, maximum. As Figure 8a depicts, for the row being increased, this entails moving along a ridge or path such as that indicated by points 827, 829, 831, 833, 835, 837, and 839. (Point 831 shows an orthogonal crossing with contour line 851.) For the row or rows being decreased, this entails either moving along a similar ridge or path but in the opposite direction, or moving parallel to an axis, e.g., from a point such as point 807 to a point such as point 809.

As the Ridge-walk process proceeds, Direct-put allocations are also increased to raise the planar portion of the surface depicted in Figure 8A.

Initialization

Initialization process 701 is shown in detail in Figure 9 and consists of the following steps:

1. In Box 901, for each resource/row of the Database 101 Resource Table, load each *availQuant* into an element of vector *resQuant*. The first row's *availQuant* goes into

resQuant[0], etc. For each of the *mProd* products/rows of the Product Table, load *potentialDemand* into the first *mProd* elements of the vector *potentialDemand*.

2. In Box 903, join Database 101 tables Group and Group Association, using *groupName* for the join. For each row of joined table, place either a group head or group element in the ***rcMat*** matrix: *productName* determines the row; *resourceName* determines the column. Place a group head in ***rcMat*** the first time each *groupName* is encountered; place a group element in ***rcMat*** each subsequent time a *groupName* is encountered. Load each group head with *atoeFnPt* structure data.

3. In Box 905, place Direct-put groups: place group heads along the diagonal of ***rcMat[mProd][0]*** through ***rcMat[m-1][nRes-1]***. For these heads, set *atoeFnPt[0].allocation* and *atoeFnPt[0].effectiveness* equal to 0; set *atoeFnPt[1].allocation* and *atoeFnPt[1].effectiveness* equal to the same very large value. Place ones (1.0) in elements *mProd* through *m-1* of the *potentialDemand* vector.

4. In Box 907, for each column of ***rcMat***, apportion the *resQuant* quantity to each of the group heads, i.e.,

```
for (j = 0; j < nRes; j++)
   for (i= each group head in column j)
      set rcMat[i][j].allocation = resQuant[j]/(number of group heads in
         column j of rcMat)
```

5. In Box 909, iterate through each column of ***rcMat*** and each element of the enumerated column that contains a group head. In other words, iterate through all group heads of ***rcMat***. For each group head,

```
if (atoeFnPt[nir].allocation < allocation)
   set dedaSub = 0
   set dedaAdd = 0
   set effectiveness = atoeFnPt[nir].effectiveness
   set maxSub = allocation - atoeFnPt[nir].allocation
   set maxAdd = 0
else
   find ir such that:
      atoeFnPt[ir].allocation <= allocation and
      atoeFnPt[ir+1].allocation > allocation
      (Conceptually, atoeFnPt[nir+1].allocation, if it existed, would be
         infinity and atoeFnPt[nir+1].effectiveness would be
         atoeFnPt[nir].effectiveness.)
```

```
if (ir < nir)
  set dedaAdd = the slope of line segment ir, i.e., the line
    determined by points atoeFnPt[ir] and atoeFnPt[ir+1]
  set maxAdd = atoeFnPt[ir+1].allocation - allocation
else
  set dedaAdd = 0
  set maxAdd = 0
if (atoeFnPt[ir].allocation not = allocation)
  set dedaSub = dedaAdd
  set maxSub =  allocation - atoeFnPt[ir].allocation
else
  if (ir not = 0)
    set dedaSub = the slope of line segment ir-1
    set maxSub =  allocation - atoeFnPt[ir-1].allocation
  else
    set dedaSub = BIG_M
    set maxSub = 0
set effectiveness = atoeFnPt[ir].effectiveness +
  dedaAdd * (allocation - atoeFnPt[ir].allocation)
set each group element effectiveness = group head effectiveness
```

(*BIG_M* is an extremely large positive number. It should be set greater than any conceivable relevant applicable number generated by this invention.)

6. In Box 911,

```
for (i = 0; i < m; i++)
  if (group heads or elements exist in row i of rcMat)
    set rowEffectiveness[i] = mathematical product of the
      effectivenesses of each group head or group element in row i
  else
    set rowEffectiveness[i] = 1
  set bOrg[i] = rowEffectiveness[i] * potentialDemand[i]
```

7. In Box 913,

```
clear a, b, c, d
set B as an identity matrix
Place ones along diagonal a[0][m] through a[mProd-1][mn-1] of matrix a.
For each row of the UnitReq table, set the appropriate element in matrix
  a equal to the value of reqQt: the field resourceName determines the
  appropriate row, with the first resource of the Resource Table
  corresponding to row mProd; productName determines the column, with
  the first product of the Product Table corresponding to column m.
set (vector) b = (vector) bOrg
set c[m] through c[mn-1] = prices of the mProd products as indicated in
  the Product Table of Database 101
```

8. In Box 915,

```
set all elements of matrix dpTie = -1
for (jProd = 0; jProd < mProd; jProd++)
  for (i = mProd; i < m; i++)
    if (0 < a[i][m+jProd])
```

```
      set dpTie[jProd][i- mProd] = i
set rwiRow = -1
For each group element (including group heads) in rcMat
   set subBlk = FALSE;
```

Initial Linear Programming Process

Once Initialization process 701 is completed, process 703 calls the LPP to maximize the formulated linear programming problem.

Axis-walk Process

Axis-walk process 705 is shown in Figure 10, and entails the following steps:

1. In Box 1001, iterate through each column of ***rcMat*** and each element of the enumerated column that contains a group head. For each group under consideration:

```
for (i = rcMat row of each group element, including the group head)
   while found (find ii such that:
        • b[ii] = 0
        • B[ii][i] > 0
        • there exists a jj such that:
             c[jj] < 0 and a[ii][jj] < 0)
          if (ii found)
             Pivot row ii as described below in Box 1117
   endwhile
   set emcSub = - c[i] * (bOrg[i]/effectiveness) * dedaSub
   if ((ir = 0 and allocation = 0) or subBlk)
      set emcSub = BIG_M
   while found (find ii such that:
        • b[ii] = 0
        • B[ii][i] < 0
        • there exists a jj such that:
             c[jj] < 0 and a[ii][jj] < 0)
          if (ii found)
             Pivot row ii as described below in Box 1117
   endwhile
   set emvAdd = - c[i] * (bOrg[i]/effectiveness) * dedaAdd
   if (ir = nir)
      set emvAdd = 0
set gmcSub = sum of the emcSub values for each group element
set gmvAdd = sum of the emvAdd values for each group element
```

2. In Box 1003, find the two groups that maximize *rcMat[ia][j].gmvAdd* minus *rcMat[is][j].gmcSub*, where *j* ranges from 0 to *nRes-1*, and *ia* and *is* reference group heads

in column *j* of ***rcMat***. Exclude from consideration groups that have elements in row *rwiRow* of ***rcMat***.

3. In Diamond 1005, test whether an allocation shift from group ***rcMat[is][j]*** to group ***rcMat[ia][j]*** is worthwhile.

4. In Box 1007, shift allocation as shown in Figure 11 and explained below.

Axis-walk Allocation Shift

Figures 11A and 11B show an enlargement of Box 1007, which entails the following steps. Steps 6 through 9 define a Box 1151.

1. In Box 1101,

```
set vector bHold = vector b
set rcMat[is][j].allocationHold = rcMat[is][j].allocation
set rcMat[ia][j].allocationHold = rcMat[ia][j].allocation
```

2. In Box 1103,

```
set awQuant = minimum(rcMat[is][j].maxSub, rcMat[ia][j].maxAdd)
```

3. In Box 1105,

```
set rcMat[is][j].allocation = rcMat[is][j].allocationHold - awQuant
set rcMat[ia][j].allocation = rcMat[ia][j].allocationHold + awQuant
```

4. In Box 1107, apply Box 909 to groups ***rcMat[is][j]*** and ***rcMat[ia][j]*** to generate group effectivenesses.

5. In Box 1109, apply Box 911 to generate *bOrg*.

6. In Box 1111, set vector *b* equal to the product of matrix ***B*** and vector *bOrg*.

7. In Box 1113, if possible, find *i* such that:

   - *b[i]* is minimized,
   - *b[i]* < 0, and
   - *bHold[i]* = 0.

8. In Diamond 1115, test whether an *i* was found in Box 1113.

9. In Box 1117, pivot row *i* as described immediately below, then go to Box 1111.

```
set irow = row to be pivoted
Find jcol such that
```

```
  • a[irow][jcol] < 0
  • c[jcol] < 0
  • c[jcol]/a[irow][jcol] is minimized
if (jcol found)
  apply prior art to pivot the simplex tableau (matrix a, vectors b and
    c, and scalar d) using a[irow][jcol] as the pivot element
```

10. In Diamond 1119, test whether any element of vector *b* is less than 0.

11. In Box 1121,

```
Find i, such that
  b[i] < 0 and
  bHold[i]/(bHold[i]-b[i]) is minimized
set awQuant = awQuant * bHold[i]/(bHold[i]-b[i])
Generate vector b by reapplying Boxes 1105, 1107, 1109, and 1111
```

(Because an infinite loop may occur in Box 1151, a limit to the number of times branching from Diamond 1115 to Box 1117 is required. Once this limit is reached, Box 1151 should be exited. If Box 1151 was entered as a result of an Axis-walk, Top-walk, or Lateral-walk call, then the ***rcMat[is][j]*** and ***rcMat[ia][j]*** pair that led to the infinite loop should be directionally blocked so as to prevent a re-entrance into Box 1151. (Directional blocking is explained as part of the Top-walk process.))

Top-walk Process

The Top-walk process considers shifting allocations from every group to every other group in each ***rcMat*** column. Because of inherent numerical accuracy limitations on most computers, it is necessary to test whether a Top-walk shift actually increased $|d|$, and if not, reverse the shift and block the considered group-pair shift possibility from further consideration. Such blocking can be accomplished by use of a three dimensional array of size *mProd* by *mProd* by ***nRes***. The first index is the ***rcMat*** row of the subtraction group-head; the second index is the ***rcMat*** row of the addition group-head; and the third index is the ***rcMat*** column of the two group heads. Initially all elements of this array are set to 0; when a group pair is blocked, the appropriate element in the array is set to 1.0. Blocking is directional.

Also, because of numerical accuracy limitations, essentially a single Top-walk shift may be accomplished by many, similar, infinitesimally-small shifts; to avoid such a possibility and the associated "waste" of CPU cycles, a minimum shifting tolerance can be used. This tolerance (*twQuantMin*) needs to be set to a non-negative value. The smaller the value of *twQuantMin*, the more accurate the solution, but the more CPU cycles required.

Top-walk works with a chain of group heads, many of which are paired into uv pairs. For each pair, the u-group has its allocation increasing and the v-group has its allocation decreasing. In Figure 8B, for example, for the 817-819 pair, group 817 is the u-group while group 819 is the v-group. Similarly for the 823-825 pair, group 823 is the u-group and 825 the v-group.

Top-walk process 707 is shown in Figure 12, and entails the following steps:

1. In Box 1201, clear all group-pair blocking for all ***rcMat*** columns.
2. In Box 1203,

```
apply Box 1001
for each group element in row rwiRow of rcMat
   set emcSub = BIG_M
   set emvAdd = -BIG_M
   in element's group head
      set gmcSub = BIG_M
      set gmvAdd = -BIG_M
for (each group head in rcMat)
   set twmcSub = gmcSub
   set twcCol = -1
   set twcRow = -1
   set twcsRow = -1
set reCycle = TRUE
while (reCycle)
   set reCycle = FALSE
   for (irow = 0; irow < mProd; irow++)
      if (b[irow] = 0 or irow = rwiRow)
         for (jcolu = 0; jcolu < nRes; jcolu++)
            if (rcMat[irow][jcolu] is a group head or group element)
               set irowuh = group-head row index of the group that has an
                  element at rcMat[irow][jcolu]
               if (rcMat[irowuh][jcolu].ir not = nir)
                  find the group head in column jcolu that has the minimum
                     twmcSub value, that has a positive allocation, and that
                     is not rcMat[irowuh][jcolu]; set irowcs = the row index
                     of the found group head
                  for (jcolv = 0; jcolv < nRes; jcolv++)
```

```
if (rcMat[irow][jcolv] is a group head or element and
   jcolu not = jcolv)
      set irowvh = group-head row index of the group
         that has an element at rcMat[irow][jcolv]
      if (rcMat[irowvh][jcolv].allocation not = 0)
         set lkqt = TWufvEpsilon(
            rcMat[irowuh][jcolu],
            rcMat[irowuh][jcolu].allocation,
            rcMat[irowvh][jcolv],
            rcMat[irowvh][jcolv].allocation)
         set mc = rc[irowcs][jcolu].twmcSub * lkqt
         for (i = each rcMat row of group
            rcMat[irowuh][jcolu])
                  if (rcMat[i][jcolv] is not an element of
                     group rcMat[irowvh][jcolv])
                          set mc = mc -
                             rcMat[i][jcolu].emvAdd * lkqt
         for (i = each rcMat row of group
            rcMat[irowvh][jcolv])
                  if (rcMat[i][jcolu] is not an element of
                     group rcMat[irowuh][jcolu])
                          set mc = mc +
                             rcMat[i][jcolv].emcSub
         if (mc < rcMat[irowvh][jcolv].twmcSub)
            set rcMat[irowvh][jcolv].twmcSub = mc
            set rcMat[irowvh][jcolv].twcRow = irowuh
            set rcMat[irowvh][jcolv].twcCol = jcolu
            set rcMat[irowvh][jcolv].twcsRow = irowcs
            set reCycle = TRUE
```

3. In Box 1205,

```
find the group pair that maximizes:
     rcMat[ia][j].gmvAdd - rcMat[is][j].twmcSub,
such that:
     • j ranges from 0 to nRes-1,
     • ia and is reference group heads in column j of rcMat,
     • the group-pair with the subtraction head at rcMat[is][j] and
       addition head at rcMat[ia][j] is not blocked
```

4. In Diamond 1207, test whether an allocation shift from group ***rcMat[is][j]*** to group ***rcMat[ia][j]*** is possibly worthwhile.

5. In Diamond 1209, test whether a transfer chain would have more than a single link. Specifically,

```
if (rcMat[is][j].twcCol = -1) then
   chain has only one link.
```

6. In Box 1211, construct a chain for shifting allocations as follows:

```
set twpGroupSub[0] = address of rcMat[is][j]
set twpGroupAdd[0] = address of rcMat[ia][j]
```

```
set twnLink = 1
set xj = j
set xis = is
set xia = ia
set crossOver = FALSE
while (not crossOver and rcMat[xis][xj].twcCol not = -1)
  set xj = rcMat[xis][j].twcCol
  set xia = rcMat[xis][j].twcRow
  set xis = rcMat[xis][j].twcsRow
  set twpGroupSub[twnLink] = address of rcMat[xis][xj]
  set twpGroupAdd[twnLink] = address of rcMat[xia][xj]
  for (i = 0; i < twnLink; i++)
    if (twpGroupSub[i] = twpGroupSub[twnLink] or
      twpGroupSub[i] = twpGroupAdd[twnLink] or
      twpGroupAdd[i] = twpGroupSub[twnLink] or
      twpGroupAdd[i] = twpGroupAdd[twnLink])
        set crossOver = TRUE
  set twnLink = twnLink + 1
set iSplitVer = -1
set iSplitHor = -1
if (crossOver)
  for (i = 0; i < twnLink - 1; i++)
    {
    if (twpGroupAdd[i] = twpGroupAdd[twnLink - 1])
      set twnLink = twnLink - 1
      goto endLoop1
    else if (twpGroupSub[i] = twpGroupSub[twnLink - 1])
      set iSplitVer = i
      goto endLoop1
    else if (twpGroupAdd[i] = twpGroupSub[twnLink - 1])
      {
      if (twpGroupSub[i] not = twpGroupAdd[twnLink - 1])
        twpGroupSub[twnLink - 1] = twpGroupSub[i]
        set iSplitVer = i
        goto endLoop1
      else
        set twnLink = twnLink - 1
        goto endLoop1
      }
    else if (twpGroupSub[i] = twpGroupAdd[twnLink - 1])
      set twnLink = twnLink - 1
      goto endLoop1
    }
  endLoop1:
for (i = 0; i < twnLink-1; i++)
  if (exactly one of the following is true:
    • CrossHAT(twpGroupSub[i])
    • CrossHAT(twpGroupAdd[i+1]))
    goto Box 1217
if (CrossHAT (twpGroupSub[twnLink-1]) and iSplitVer = -1)
  {
  for (i = 0; i < twnLink; i++)
    if (CrossHAT(twpGroupSub[i]))
      {
      if (iSplitHor = -1)
```

```
            set iSplitHor = i + 1
         else
            set twnLink = i + 1
            goto endLoop2
         }
   goto Box 1217
   }
endLoop2:
```

Function definition:

```
CrossHAT(pointer group head (pGH))
   if (the group whose head is pointed to by pGH has an element in row
      rwiRow of rcMat)
         return TRUE
   else
         return FALSE
```

7. In Box 1213, determine quantities and shift allocations through the chain. This is shown in detail Figure 13 and explained below.
8. In Diamond 1215, test whether the allocation shifts through the chain proved worthwhile.
9. In Box 1217, block the shift group-pair with a subtraction head at ***rcMat[is][j]*** and an addition head at ***rcMat[ia][j]*** (both group heads were determined in Box 1205) from further consideration.
10. In Box 1219, apply Box 705 (Axis-walk).
11. In Diamond 1221, test whether $|d|$ has increased since any group-pair was blocked in Box 1217.

Top-walk Allocation Shift

Figures 13A, 13B, and 13C show Box 1213 in detail:

1. In Box 1301, save the following to a temporary memory location that is specific to this Top-walk process:
   - matrix ***a***, vectors *b* and *c*, and scalar *d*
   - matrix ***rcMat*** and all contained group head and group elements
   - vectors *bOrg* and *rowEffectiveness*
2. In Box 1302,

```
set vector bHold = vector b
```

```
for (i = 0; i < twnLink; i++)
  apply to group pointed to by twpGroupSub[i]
    set allocationHold = allocation
    set effectivenessHold = effectiveness
  apply to group pointed to by twpGroupAdd[i]
    set allocationHold = allocation
    set effectivenessHold = effectiveness
```

3. In Box 1303, set *twQuant*, the initial shift quantity:

```
set twQuant = twpGroupAdd[0]->maxAdd
for (i = 0; i < twnLink-1; i++)
  set twQuant = minimum (twQuant, twpGroupSub[i]->maxSub)
  set twQuant = TWufv(twpGroupAdd[i+1],
                      twpGroupAdd[i+1]->allocation,
                      twpGroupSub[i],
                      twpGroupSub[i]->allocation,
                      twQuant)
  set twQuant = minimum (twQuant, twpGroupAdd[i+1]->maxAdd)
set twQuant = minimum (twQuant, twpGroupSub[twnLink-1]->maxSub)
```

The following functions are used in Box 1303 and in other Boxes of the Top-walk process. *TWufv* accepts a quantity being shifted out of a group v and determines the compensating quantity to shift into a group u; *TWvfu* does the reverse. *TWufvEpsilon* is the same as *TWufv*, except the quantity being shifted out of group v, in the mathematical limit sense, is assumed to be an infinitesimally small unit of one, while the compensatory quantity shifted into group u is a multiple of the same infinitesimally small unit.

```
GenEffectiveness(pointerGroup, newAllocation)
  set net = pointerGroup->effectivenessHold
  set diff = newAllocation - pointerGroup->allocationHold
  if (0 < diff)
    set net = net + pointerGroup->dedaAdd * diff
  else
    set net = net + pointerGroup->dedaSub * diff
  return net
TWufv(pointerUGroup, uAllocation, pointerVGroup, vAllocation, shift)
  set ue = GenEffectiveness(pointerUGroup, uAllocation)
  set ud = pointerUGroup->dedaAdd
  set ve = GenEffectiveness(pointerVGroup, vAllocation)
  set vd = pointerVGroup->dedaSub
  set vi = vd * shift
  return (ue * vi/(ud * (ve - vi)))
TWvfu(pointerUGroup, uAllocation, pointerVGroup, vAllocation, shift)
  set ue = GenEffectiveness(pointerUGroup, uAllocation)
  set ud = pointerUGroup->dedaAdd
  set ve = GenEffectiveness(pointerVGroup, vAllocation)
  set vd = pointerVGroup->dedaSub
  set ui = ud * shift
  return (ve * ui/(vd * (ue + ui)))
```

```
TWufvEpsilon(pointerUGroup, uAllocation, pointerVGroup, vAllocation)
   set ue = GenEffectiveness(pointerUGroup, uAllocation)
   set ud = pointerUGroup->dedaAdd
   set ve = GenEffectiveness(pointerVGroup, vAllocation)
   set vd = pointerVGroup->dedaSub
   return (ue*vd/ud*ve)
TWvfuEpsilon(pointerUGroup, uAllocation, pointerVGroup, vAllocation)
   set ue = GenEffectiveness(pointerUGroup, uAllocation)
   set ud = pointerUGroup->dedaAdd
   set ve = GenEffectiveness(pointerVGroup, vAllocation)
   set vd = pointerVGroup->dedaSub
   return (ud*ve/ue*vd)
```

4. In Box 1305, shift allocations as follows:

```
set shift = twQuant
for (i = twnLink-1; 0 <= i; i--)
   set twpGroupSub[i]->allocation =
      twpGroupSub[i]->allocationHold - shift
   if (i = iSplitVer)
      set shift = shift - twQuant
   set twpGroupAdd[i]->allocation =
      twpGroupAdd[i]->allocationHold + shift
   if (i = iSplitHor)
      set debt = TWufv(   twpGroupAdd[iSplitHor],
                          twpGroupAdd[iSplitHor]->allocation,
                          twpGroupSub[twnLink-1],
                          twpGroupSub[twnLink-1]->allocation,
                          twQuant)
      set shift = shift - debt
   else
      set debt = 0
   if (0 < i)
      set shift = TWvfu(  twpGroupAdd[i],
                          twpGroupAdd[i]->allocationHold+debt,
                          twpGroupSub[i-1],
                          twpGroupSub[i-1]->allocationHold,
                          shift)
   generate group effectivenesses for the groups pointed to by
      twpGroupSub[i] and twpGroupAdd[i] by applying Box 909
regenerate vectors rowEffectiveness and bOrg by applying Box 911
```

5. In Box 1307, apply Box 1001 to generate group marginal values for each group pointed to by vectors *twpGroupSub* and *twpGroupAdd*. (Note that the linear programming problem and solution is the same as it was in Box 1301.)

6. In Box 1309, do the following to determine *rcMat[is][j].twmcSub*, given the shifts done in Box 1305:

```
set mc = twpGroupSub[twnLink-1]->gmcSub
set shift = 1.0 //(infinitesimal unit)
for (i = twnLink-1; 1 <= i; i--)
```

```
   set jj = rcMat column of group pointed to by twpGroupAdd[i]
   for (ii = each rcMat row of group pointed to by twpGroupAdd[i])
     if (group pointed to by twpGroupSub[i-1] does not have group
       element in row ii of rcMat)
         set mc = mc - rcMat[ii][jj].emvAdd * shift
   if (i = iSplitVer)
     set shift = shift - 1.0
   if (i = iSplitHor)
     set debt = TWufvEpsilon(twpGroupAdd[iSplitHor],
                             twpGroupAdd[iSplitHor]->allocation,
                             twpGroupSub[twnLink-1],
                             twpGroupSub[twnLink-1]->allocation)
     set shift = shift - debt
   set shift = shift * TWufvEpsilon(twpGroupAdd[i],
                             twpGroupAdd[i]->allocation,
                             twpGroupSub[i-1],
                             twpGroupSub[i-1]->allocation)
   set jj = rcMat column of group pointed to by twpGroupSub[i-1]
   for (ii = each rcMat row of group pointed to by twpGroupSub[i-1])
       if (group pointed to by twpGroupAdd[i] does not have group
         element in row ii of rcMat)
           set mc = mc + rcMat[ii][jj].emcSub * shift
 set rcMat[is][jj].twmcSub = mc
```

7. In Diamond 1311, test whether the shifting done in Box 1305 is marginally worthwhile, i.e., whether, ***rcMat[is][j].twmcSub*** <= ***rcMat[ia][j].gmvAdd.***

8. In Box 1315, use bisection method search to find a new value for ***twQuant*** so that:
   - it is between 0 and the values set in Box 1303 and
   - after reapplying Boxes 1305, 1307, and 1309 the following condition is met:

     rcMat[is][j].twmcSub = ***rcMat[ia][j].gmvAdd***

9. In Box 1317, apply Box 1305.

10. In Box 1321, apply Box 1151 to generate vector *b*.

11. In Diamond 1329, test whether any element of vector *b* is less than an infinitesimal negative value.

12. In Box 1331, use bisection method search to find a new value for *twQuant*, so that:
    - it is between 0 and the smaller of the values as set in Boxes 1303 and 1315.
    - after reapplying Box 1317 and setting $b = B * bOrg$, the smallest element in vector *b* is 0 or infinitesimally smaller than 0.

14. In Box 1333,

```
if (twQuant < twQuantMin)
```

```
set twQuant = minimum of twQuantMin and twQuant as set in Box 1303
```

15. In Box 1335, apply Box 1305 using the current *twQuant* and set $b = B * bOrg$.
16. In Box 1337, make the current linear programming solution feasible, by, for instance, applying the well known Dual Simplex Method.
17. In Diamond 1339, test whether $|d|$ has increased since it was saved in Box 1301.
18. In Box 1341, restore the earlier solution by restoring the data saved in Box 1301.

Lateral-walk Process

Lateral-walk process 709 uses *facReduce* as a programmer-set tolerance, which needs to be slightly less than 1.0. The closer *facReduce* is to 1.0, the more accurate the solution, but the more CPU cycles required. Like the Top-walk process, the Lateral-walk process tracks which group-pair shifts proved undesirable and then avoids repeat consideration of such shifts. Process 709 is shown in detail in Figure 14 and entails the following steps:

1. In Box 1401, clear all group-pair blockings.
2. In Box 1403, apply Box 1301, but use storage that is specific to this Lateral-walk process. Also make a copy of vector *potentialDemand.*
3. In Box 1405,

```
for (i = 0; i < m; i++)
  set limitLoop = a positive integer limit value
  while (b[i] = 0 and 0 < limitLoop and
      (exists j and jj such that
       B[i][j] not = 0
       B[i][jj] not = 0
       j not = jj))
         {
         set potentialDemand[i] = potentialDemand[i] * facReduce
         apply Box 911
         set b = B * bOrg
         apply box 1337
         set limitLoop = limitLoop - 1
         }
```

4. In Box 1407, apply Boxes 1203, 1205, 1207, 1209, 1211, 1213, and 1219. Exit before applying Boxes 1215 and 1221. When doing Box 1205, respect any pair-blocking done in Box 1419. When doing Box 1213, skip Diamond 1339 and Box 1341. Immediately exit Box 1219, after doing Box 1007.
5. In Box 1409, restore vector *potentialDemand* that was stored in Box 1403. Also

```
apply Box 911
set b = B * bOrg
apply box 1337
```

6. In Diamond 1411, test whether a Top-walk allocation shift was done in Box 1407 i.e., if the answer to the condition of Box 1207 was yes.
7. In Diamond 1415, test whether |*d*| increased from its value saved in Box 1403.
8. In Box 1417, restore the solution saved in Box 1403.
9. In Box 1419, block the group-pair with group heads at ***rcMat[is][j]*** and ***rcMat[ia][j]*** (as determined in Boxes 1407 and 1205) from further consideration.
10. In Diamond 1421, test whether |*d*| has increased since any group-pair was blocked in Box 1419.

Ridge-walk Process

Ridge-walk process 711 uses three programmer-set tolerances: *rwATLrefresh*, *rwShiftMin*, and *rwShiftMax*. These tolerances need to be positive. Once the increase in *rowEffectiveness* is greater than *rwATLrefresh*, the Axis-walk, Top-walk, and Lateral-walk processes are called. Tolerances *rwShiftMin* and *rwShiftMax*, with *rwShiftMin* <= *rwShiftMax*, determine the minimum and maximum allocation shift per iteration. The smaller each of these three tolerances, the more accurate the solution, but the more CPU cycles required.

Ridge-walk process 711 is shown in detail in Figure 15 and entails the following steps:

1. In Box 1501, use *rwiRow* as an iterator to continually cycle through the first *mProd* rows of ***rcMat***. Continue until a complete cycle has not resulted in any increase in |*d*|. Specifically:

```
set rwiRow =0
```

```
set count = 0
do
   {
   set dHold = |d|
   apply Box 1551
   if (|d| > dHold)
      set count = 1
   else
      set count = count + 1
   set rwiRow = rwiRow + 1
   if (rwiRow = mProd)
      set rwiRow = 0
   }
while (count not = mProd)
set rwiRow = -1
```

2. In Box 1503,

```
set all elements of vector dpTieSubBlk = FALSE
set baseRowEffectiveness = - BIG_M
```

3. In Box 1505, apply Box 1301, but use storage that is specific to this Ridge-walk process.

4. In Box 1507, drag along Direct-puts: shift group allocations between the groups of row *rwiRow* and its Direct-put groups in order to relieve constraints on product *rwiRow*. Specifically,

```
for (j = 0; j < nRes; j++)
   if (dpTie[rwiRow][j] not = -1)
      if (rcMat[rwiRow][j] is not empty)
         set iRW = row of group head of the group that has an element at
            rcMat[rwiRow][j]
      else
         set iRW = -1
      set iDP = dpTie[rwiRow][j]
      set qtRW = bOrg[rwiRow]
      set qtDP = (bOrg[iDP]) /(the value of a[iDP][m + rwiRow] as
         originally set in Box 913)
      while (qtDP < qtRW)
         apply Box 1001 to all groups in column j of rcMat
         ia = iDP
         is = index of group head in column j of rcMat that has the smallest
            gmcSub but is not equal to iDP
         if (rcMat[is][j].gmcSub = BIG_M)
            break out of while loop
         set awQuant = minimum ( rcMat[ia][j].maxAdd,
                                 rcMat[is][j].maxSub,
                                 rwShiftMin)
         apply Boxes 1101, 1105, 1107, 1109, 1111, and 1337
         if (is = iRW)
            set dpTieSubBlk[j] = TRUE
         set qtRW = bOrg[rwiRow]
```

```
    set qtDP = bOrg[iDP] / (the value of a[iDP][m + rwiRow] as
      originally set in Box 913)
  if (iRW not = -1)
    set is = iDP
    set ia = iRW
    do
      apply Box 1001 to groups rcMat[is][j] and rcMat[ia][j]
      if (Diamond 1005 is TRUE)
        apply Box 1007
    while (Diamond 1005 is TRUE)
```

5. In Diamond 1509, test whether *rowEffectiveness[rwiRow]* exceeds *baseRowEffectiveness* plus *rwATLrefresh*.

6. In Box 1511,

```
for (j = 0; j < nRes; j++)
  if (rcMat[rwiRow][j] is not empty)
    set rcMat[rwiRow][j].subBlk = TRUE
  if (dpTie[rwiRow][j] not = -1)
    set rcMat[dpTie[rwiRow][j]][j].subBlk = TRUE
apply the following:
  Axis-walk (Box 705)
  Top-walk (Box 707)
  Lateral-walk (Box 709)
for (j = 0; j < nRes; j++)
  if (rcMat[rwiRow][j] is not empty)
    set rcMat[rwiRow][j].subBlk = FALSE
  if (dpTie[rwiRow][j] not = -1)
    set rcMat[dpTie[rwiRow][j]][j].subBlk = FALSE
set baseRowEffectiveness = rowEffectiveness[rwiRow]
```

7. In Diamond 1513, test whether $|d|$ is greater than the last value of $|d|$ stored in Box 1505 or 1515.

8. In Box 1515, apply Box 1505.

9. In Box 1517, attempt Ridge-walk iteration, which is explained in detail below.

10. In Diamond 1519, test whether a Ridge-walk iteration was done in Box 1517.

11. In Box 1521, restore the solution last saved in Boxes 1505 and 1515.

Ridge-walk Iteration

Ridge-walk iteration 1517 is shown in detail in Figures 16A and 16B.

1. In Box 1601,

```
set applied1007 = FALSE
set all elements of rwpDest and rwpSour equal to NULL
```

```
for (j = 0; j < nRes; j++)
  set loopRepeat = TRUE
  while (loopRepeat and exist group element at rcMat[rwiRow][j])
    set loopRepeat = FALSE
    set rwpDest[j] = address of group head of the group having an
      element at rc[rwiRow][j]
    if (rwpDest[j]->ir = rwpDest[j]->nir)
        exit while loop
    apply Box 1001
    Attempt to find group head in column j such that:
      • gmcSub is minimized
      • the group head is not pointed to by rwpDest[j]
      • the group head has an allocation greater than 0
      • if dpTieSubBlk[j] is TRUE, then the group is not
        rcMat[dpTie[rwiRow][j]][j]
    if (group head is found)
      {
      set rwpSour[j] = address of found group head
      if (rwpSour[j]->gmcSub < rwpDest[j]->gmvAdd)
        set ia = row of group head rwpDest[j]
        set is = row of group head rwpSour[j]
        apply Box 1007
        set applied1007 = TRUE
        set loopRepeat = TRUE
      }
    else
      set rwpSour[j] = NULL
if (applied1007)
  goto Box 1507, i.e. exit Fig. 16 and assume an iteration
```

2. In Diamond 1603, test whether there exists a *jj*, such that both *rwpDest[jj]* and *rwpSour[jj]* are not NULL. If such a *jj* exists, then a Ridge-walk iteration is possible. The iteration will simultaneously apply to each non-null *rwpDest[jj]-rwpSour[jj]* pair. (To facilitate exposition, all elements of vectors *rwpDest* and *rwpSour* will be assumed to be non NULL.)

3. In Box 1605,

```
for (each group head pointed to by vectors rwpDest and rwpDest)
  set allocationHold = allocation
```

4. In Box 1607,

```
set vector bHold = vector b
for (j = 0; j < nRes; j++)
  set rwOldAlloc[j] =
    rwpDest[j]->effectiveness / rwpDest[j]->dedaAdd
  set rwOldMC[j] = rwpSour[j]->gmcSub
```

5. In Box 1609,

```
set rwParaMin = BIG_M
set rwParaMax = BIG_M
```

```
for(j = 0; j < nRes; j++)
   set min = minimum (rwpDest[j]->maxAdd,
                      rwpSour[j]->maxSub,
                      rwShiftMin)
   set min = (min + rwOldAlloc[j]) * rwOldMC[j]
   set rwParaMin = minimum(min, rwParaMin)
   set max = minimum (rwpDest[j]->maxAdd,
                      rwpSour[j]->maxSub,
                      rwShiftMax)
   set max = (max + rwOldAlloc[j]) * rwOldMC[j]
   set rwParaMax = minimum(max, rwParaMax)
set rwParameter = rwParaMax
```

6. In Box 1611,

```
for (j = 0; j < nRes; j++)
   set shift = rwParameter/rwOldMC[j] - rwOldAlloc[j]
   if (shift < 0)
      set shift = 0
   set rwpSour[j]->allocation =
      rwpSour[j]->allocationHold - shift
   set rwpDest[j]->allocation =
      rwpDest[j]->allocationHold + shift
apply Box 909 to groups pointed to by vectors rwpSour and rwpDest
```

7. In Box 1613, generate *bOrg* by applying Box 911.

8. In Box 1621, apply Box 1321.

9. In Diamond 1623, test whether any element of vector *b* is less than an infinitesimal negative value.

10. In Box 1625, use bisection method search to find a new value for *rwParameter*, so that:

    - it is between 0 and *rwParaMax*
    - after applying Boxes 1611 and 1613, and setting $b = B * bOrg$, the smallest element in vector *b* is 0 or infinitesimally smaller than 0.

11. In Box 1627,

```
if (rwParameter < rwParaMin)
   set rwParameter = rwParaMin
```

12. In Box 1629, apply Boxes 1611 and 1613 using the current *rwParameter* and set $b = B * bOrg$.

13. In Box 1631, as in Box 1337, make the current linear programming solution feasible.

Finalization

The finalization process of posting the results to the database (process 713) is shown in Figure 17 and entails:

1. In Box 1701, generate marginal values by applying Box 1001.

2. In Box 1703,

```
for (j = 0; j < nRes; j++)
  do
    set is = index of group head in column j of rcMat that has the
      smallest gmcSub, such that is < mProd
    if (rcMat[is][j].gmcSub = 0)
      set ia = mProd + j
      apply Box 1007, then Box 1001
  while (rcMat[is][j].gmcSub = 0)
  set meanUse field in Database 101 Resource Table = (sum of the
    allocations to all the group heads in column j and rows 0 through
    row mProd-1 of rcMat) + (the quantity of the resource in row (mProd
    + j) of matrix a and vector b allocated by the LPP)
  set marginalValue = the minimum value of gmcSub contained in all the
    group heads in column j of rcMat
```

3. In Box 1705,

```
for (each group head in the first mProd rows of rcMat)
  set i = group-head rcMat row
  set j = group-head rcMat column
  Locate the row in the Group Table that corresponds to group head
    rcMat[i][j]; i.e., back trace to the original row used in Box 903
  set the meanAlloc field in the Group Table row =
    rcMat[i][j].allocation
  set the marginalValue field in the Group Table row =
    rcMat[i][j].gmcSub
```

4. In Box 1707,

```
for (iProd = 0; iProd < mProd; iProd++)
  apply Boxes 1709 through 1715 to generate data for the Product Table
```

5. In Box 1709, for row *iProd* of the Product Table, apply prior-art linear programming methods to set *meanSupply* equal to quantity of *iProd* produced.

6. In Diamond 1711, test whether *meanSupply* as set in Box 1709 equals 0.

7. In Box 1713, set *marginalCost* = *price*.

8. In Box 1715,

```
set mmc = 0
set rwiRow = iProd
if (bOrg[iProd] < 1.0)
```

```
    apply Box 1301, but use storage specific to this Box
    apply Box 1601, but without branching to Box 1507
    apply Diamond 1603
    if (iteration not possible as per Diamond 1603)
      set marginalCost (of row iProd of Product Table) = infinity
      exit this Box
    apply Boxes 1605 and 1607
    set rwParaMin = 0
    set rwParaMax = BIG_M
    set rwParameter = BIG_M
    Use bisection search method to find rwParameter value so that, after
      applying Boxes 1611 and 1613, bOrg[iProd] equals 1.  If this is not
      possible, continue with bisection to find rwParameter that
      maximizes bOrg[iProd].
    for (j = 0; j < nRes; j++)
      set mmc = mmc +  (rwpDest[j]->allocation -
                        rwpDest[j]->allocationHold) *
                       (rwOldMC[j]- (rwpDest[j]->gmvAdd -
                        rcMat[iProd][j].emvAdd))
                       //if rcMat[iProd][j] is empty, assume
                       //rcMat[iProd][j].emvAdd = 0

    if (bOrg[iProd = 0)
      set marginalCost (of row iProd of Product Table) = infinity
      exit this Box
    set mmc = mmc / bOrg[iProd]
    apply Box 1341, but use storage specific to the Box
  Join the Resource and UnitReq Tables using resourceName for the join-key
    and only include rows of the UnitReq Table that correspond to product
    iProd.
  for (each row of the joined table)
    set mmc = mmc + marginalValue * reqQt
  set marginalCost (of iProd in Product Table) = mmc + sumWICash[iProd]
  set rwiRow = -1
```

<u>Detailed Description Preferred Embodiment</u>

The preferred embodiment builds upon the previously described basic embodiment and makes possible all the previously described objects and advantages. It entails enhancements to the database, handling of cash related resources, Monte Carlo simulation, operation under a GUI (Graphical User Interface), optimization controls, and generating supply and demand schedules that facilitate analysis.

When Monte Carlo simulation is done, the following, which is here termed a scenario, is repeated: *potentialDemand* values are randomly drawn from user-defined statistical

distributions, optimized allocations are made, and the results noted. A set of scenarios constitutes what is here termed a simulation. Once a simulation is finished, mean noted-scenario-results are written to the database. Unfulfilled *potentialDemand* of one scenario is possibly passed on to the next. Each scenario is fundamentally a possibility for the same period of consideration, e.g., the upcoming month. Implicitly, a steady stochastic state is being presumed for the period of consideration. (For purposes of the present invention's making direct allocations as described in the Theory of the Invention Section, a simulation is done with only one scenario; if non-single-point statistical distributions are specified for *potentialDemand*, then mean values are used for the single scenario.)

A Base simulation is the basic simulation done to allocate resources and determine marginal costs/values. A Supply simulation generates the schedule between product price and optimal mean supply quantity. Similarly, a Demand simulation generates the schedule between external resource price and optimal quantities.

To facilitate exposition, programming objects are utilized. These are the objects of object-oriented programming, and conceptually consists of a self-contained body of data and executable code.

Database

The preferred embodiment database has two additional tables: Distribution and Results Tables. All the previous five tables have additional fields.

Distribution Table

The Distribution Table is in effect a user-defined library of statistical distributions that can be used to express Potential-demand as a statistical distribution. This table has the following fields, one of which is a programming object. Those marked with asterisks (*) are determined by the present invention:

- *distName* — user defined name; table key.
- *distType* — type of distribution, e.g., normal, uniform, Poisson, single-value, etc.
- *distObject* — a programming object that:
    1. accepts and displays distribution parameters, (for example, for a normal distribution, the mean and standard deviation).
    2. draws a graph of the specified distribution.
    3. generates random values drawn from the specified distribution.
    4. generates mean expected values (for direct allocations).
- *meanGen** — the mean of generated random values for the last executed Base simulation.
- *marginalValue** — the mean marginal value of the *potentialDemand*(s) generated by *distObject*.

Resource Table

The Resource Table has the following additional fields, each of which is set by the user:

- *unit* — e.g., liter, hour, etc.
- *availability* — either "fixed" or "buyable."
- *WTMD* — wear-and-tear market depreciation. This is the market-value depreciation resulting from using the resource. It is different from, and in contrast to, depreciation occurring solely because of the passage of time.
- *payPrice* — the full cash price that needs to be paid to obtain a buyable resource.
- *demandObject** — an object that shows a demand (marginal value) schedule and associated data.

If availability is "fixed", then *WTMD* is applicable and *payPrice* is not applicable. Conversely, if availability is "buyable," then *WTMD* is not applicable and *payPrice* is applicable. WI-cash needs to be included as a resource in the Resource Table. Its quantity is the amount of cash that is available to finance buyable resources.

Group Table

The Group Table has two additional fields. The *fixedAlloc* field indicates whether the user wishes to manually set a group allocation. If "Yes" is specified, then a fixed allocation quantity needs to be specified in the second field, *fxAlQt*. If such a manual setting is done, then the initialization process sets the allocation to *fxAlQt* and the allocation is not changed by the Axis-walk, Top-walk, Lateral-walk, or Ridge-walk processes.

Product Table

The Product Table has the following additional fields. Those marked with asterisks (*) are determined by the present invention:

- *fillValue* — the value to the organization above and beyond the price paid for the product:
  - For governments and non-profits, it is the estimated societal value of providing a unit of product (service), minus the price, if any, paid. It, plus any paid price, is a monetary, quantitative measurement of a fulfilling an organization's mission by providing a unit of product. It can be estimated subjectively or by using the techniques of welfare economics.
  - For commercial concerns, it is the expected value received beyond the paid price. This would be typically used for new products, when initially building market-share and market-size is of predominate importance. It is the value to the organization of getting customers to buy the product, besides and in addition to, the actual price paid.

  fillValue can also include the value to the organization of being able to supply a product in order to maintain its reputation as a reliable supplier.
- *distName* — the statistical distribution to be used to generate *potentialDemand* values. Joins with field of the same name in the Distribution Table.
- *distPercent* — the percent of the generated random value, from the statistical distribution, that should be used as *potentialDemand*.

- *carryOver* — the percentage of unfulfilled *potentialDemand* that carries over from one scenario to the next.
- *meanPotentialDemand** — mean scenario *potentialDemand* for the most recent Base simulation.
- *supplyObject** — an object that shows a supply (marginal cost) schedule, an average opportunity cost schedule, and associated data.

The fields *distName*, *distPercent* and *carryOver* replace the earlier *potentialDemand* field of the Product Table. They are used to generate the previously discussed *potentialDemand* vector.

UnitReq Table

The UnitReq Table has an additional field named *periodsToCash*, which is set by the user. This is the number of time periods between purchasing the resource and receiving payment for the product. This field is only applicable for resources whose availability is "buyable."

Results Table

The Results Table has fields for both accepting user-defined parameters and reporting optimization results. The latter type fields are marked below with asterisks (*) and are means of scenario results for Base simulations. Not listed, but following each field marked with an exclamation point (!), is a field that contains the standard errors of the marked field:

- *Sequence* — table key.
- *Internal Producer's Surplus**! — previously explained.
- *Change in WI-cash**! — mean of scenario-aggregate change in WI-cash.
- *WI-cash* — start amount for each scenario; same as a *availQuant* for WI-cash in Resource Table.
- *Marginal Value of WI-cash**! — mean of scenario-aggregate marginal values of WI-cash; same as a marginal value for WI-cash in Resource Table.
- *Sum Fill Value**! — mean of scenario-aggregate *fillValues*.
- *Sum WTMD**! — mean of scenario-aggregate *WTMD*.

- *Allocation* — either "Direct" or "Indirect."
- *Maximization Type* — either "IPS" or "WI-cash."
- *WI-cash Type* — either "Spread-out" or "Fold-in." Spread-out signifies that WI-cash need only finance the current period's buyable resources for the current period, i.e., the financing is spread over multiple periods and no concern about future financing is warranted. Fold-in signifies that WI-cash needs to finance the total current period's expenditure for buyable resources, i.e., all current and future financing is folded-into the current period, which WI-cash needs to cover.
- *Rand Seed* —random number generator seed.
- *N Sample*— number of scenarios per simulation.
- *MC/MV Display* — either:
    - "Partial" — meaning that simple marginal costs (*gmcSub*) should be used for reporting.
    - "Infinite Series" — meaning that Top-walk marginal costs (*twmcSub*) should be used for reporting.
    - "Quantum" — meaning that the process used to generate supply and demand schedules should be used to determine marginal costs and marginal values used for reporting.
- *Max Base RW Iterations* — times *mProd* is maximum number of times Box 1551 should be executed per base scenario.
- *Max Base RW Time* — maximum time that should be spent in Box 1551 per base scenario.
- *Max S/D RW Iterations* — times *mProd* is maximum number of times Box 1551 should be executed per supply and demand scenario.
- *Max S/D RW Time* — maximum time that should be spent in Box 1551 per supply and demand scenario.

The *Sequence* field enumerates the rows of the Results Table, with the first row having a *Sequence* value of 0. Each time a Base simulation is done, all the positive *Sequence* values are

incremented by 1; the row with a *Sequence* value of 0 is duplicated, the simulation results are stored in this duplicate row, its *Sequence* value is set to 1.

Graphical User Interface

The preferred GUI embodiment has four windows: Distributions, Resources, Products, and Results. These windows show all database data, which the user can view and edit. The statistical distributions, allocations-to-effectiveness functions, and supply and demands schedule are shown both tabularly and graphically. The data the user enters and edits is in a foreground/background color combination that differs from the foreground/background color combination of the data determined by the present invention.

These windows have state-of-the-art editing and viewing capabilities, including (without limitation) cutting-and-pasting, hiding and unhiding rows and columns, font and color changing etc. Such generic windows and generic capabilities are common for: 1) a personal computer, such as the Apple Macintosh and the systems running Microsoft Windows, and 2) computer work stations, such as those manufactured by Digital Equipment Corp., Sun Micro Systems, Hewlett-Packard, and the International Business Machines Corp.

The Distribution Table is shown in its own window. An example of such window, with column titles and sample data rows, is shown in Figure 18. (The small triangle in the figure is to adjust the bottom of the graph.)

The Resource and Group Tables are shown in their own window, with groups defined below the resources they use. An example of such a window with the first few rows is shown in Figure 19. (The empty oval signifies the compression of an empty table and graph; a solid oval signifies the compression of a table and graph containing data).

The Product and UnitReq Tables are merged together in their own window. An example of such a window is shown in Figure 20.

The Results Table is shown in its own window, as shown in Figure 21. The *Next* column is for the row of *Sequence* 0; *Current* is for *Sequence* 1; *Previous(0)* for *Sequence* 2; etc. Additional table rows are inserted as columns between the *Next* and *Previous(0)* columns, with the "oldest" immediately to the right of the *Next* column.

Base Simulation

The procedure of the preferred embodiment allocation is shown in Figure 22, which builds upon the procedure shown in Figure 7, entails the following:

1. In Box 2201,

```
for (iProd = 0; iProd < mProd; iProd++)
   set sumWICash[iProd] = 0
   Join Resource, UnitReq, and Product tables where
      • ProductTable.productName = UnitReqTable.productName
      • ResourceTable.resourceName = UnitReqTable.resourceName
      • ProductTable.productName is product iProd
   for (each row of joined table)
      if (WI-cash Type = Spread-out)
         set sumWICash[iProd] = sumWICash[iProd] + payPrice * reqQt
      else
         set sumWICash[iProd] = sumWICash[iProd] + payPrice * reqQt *
            periodsToCash
```

2. In Box 2203,

```
Clear vector c
set c[m] through c[mn-1] = prices of the mProd products as indicated in
   the Product Table of Database 101
for (iProd = m; iProd < mn; iProd++)
   set c[iProd] = c[iProd] - sumWICash[iProd-m]
   if (Maximization Type = IPS)
      set c[iProd] = c[iProd] + (fillValue for product (iProd - m))
      Join Resource and UnitReq tables where:
         • ResourceTable.resourceName = UnitReqTable.resourceName
         • ResourceTable.availability = "fixed"
         • UnitReqTable.productName is product iProd
      for (each row of joined table)
         set c[iProd] = c[iProd] - WTMD * reqQt
```

3. In Box 2205,

```
for (iScenario = 0; iScenario < N_Sample; iScenario++)
  apply Boxes 2207 through 2211
```

4. In Box 2207,

```
if (iScenario = 0)
  Use randSeed to generate random seeds for each distribution object.
Cause each distribution object to draw a random number from its
  distribution.
for (iProd = 0; iProd < mProd; iProd++)
  set potentialDemand[iProd] =
    (product's distObject's random value) * distPercent + carryOver *
      (unfulfilled potentialDemand for iProd from previous period, if
      it existed)
```

5. In Box 2209, directly apply Boxes 701 through 711, with the following exceptions:

- use the *c* vector generated in Box 2203
- exclude from matrix and vector loading all buyable resources
- include WI-cash as a fixed availability resource
- load into matrix a *sumWICash[iProd]* as product *iProd*'s requirement of WI-cash
- limit the number of times Box 1551 is executed to *baseMaxRWItertions* times *mProd*
- limit the total time spent in Box 1551 to *baseMaxRWTime* seconds

6. In Box 2211, compute and note scenario results.

```
apply Box 713, except note, rather than write, resulting data
if (MC/MV Display = "Infinite Series")
  When applying Box 713, apply Box 1203, rather than Box 1701, and set
    both gmcSub and gmvAdd equal to twmcSub for each group in rcMat.
if (MC/MV Display = "Quantum")
  for each resource
    set resourceQuant = availQuant
    apply Boxes 2403 thru 2407
    note yielded resource price, in Box 2407, as being marginal value
      of resource
  for each product
    Use bisection search method to find productPrice so that applying
      Boxes 2303 through 2309 yields an increment of 1.0 in the number
      of units produced of the considered product.  Note productPrice
      as being the marginal cost of producing the considered product.
set scenIPS = 0
set scenWICashChange = 0
set scenfillValue = 0
set scenWTMD = 0
for (each distribution object)
  set marginalValue = 0
```

```
for (iProd = 0; iProd < mProd; iProd++)
  set quant = (LPP's determined quantity for product iProd)
  set price = price of product iProd
  set fillValue = fillValue for a unit of product iProd
  set cashOut = 0
  set wtmdOut = 0
  Join Resource, UnitReq, and Product tables where
    • ProductTable.productName = UnitReqTable.productName
    • ResourceTable.resourceName = UnitReqTable.resourceName
    • ProductTable.productName is product iProd
  for (each row of joined table)
    set cashOut = cashOut + payPrice * reqQt
    set wtmdOut = wtmdOut + wtmd * reqQt
  set scenIPS = scenIPS + quant * (price + fillValue - cashOut -
    wtmdOut)
  set scenWICashChange = scenWICashChange + quant * (price - cashOut)
  set scenfillValue = scenfillValue + quant * fillValue
  set scenWTMD = scenWTMD + quant * wtmdOut
  while found (find ii such that:
      • b[ii] = 0
      • B[ii][iProd] > 0
      • there exists a jj such that:
          c[jj] < 0 and a[ii][jj] < 0)
        if (ii found)
          Pivot row ii as described in Box 1117
  endwhile
  set pDistObject = pointer to distribution object used to generate
    potentialDemand[iProd]
  set pDistObject->marginalValue =
    pDistObject->marginalValue +
    (- c[iProd] * bOrg[iProd]/potentialDemand[iProd])
```

7. In Box 2213, compute means and standard errors of *scenIPS*, *scenfillValue*, *scenWICashChange*, *scenWTMD* (of Box 2211) and update Results table. For each distribution, compute the mean of scenario *marginalValue* as calculated in Box 2211 and update distribution table. Compute means of resource and product quantities and marginal values/costs; update appropriate tables. Update GUI database display.

Supply Simulation

The procedure to generate product supply schedules is shown in Figure 23. For expository purposes, the supply schedule being generated is for a product *iProdSup* and will have prices between *lowPrice* and *highPrice* with fixed increments. This entails,

1. In Box 2301,

```
for (productPrice = lowValue; productPrice < highPrice;
```

```
productPrice = productPrice + increment)
   apply Boxes 2303 through 2309
```

2. In Box 2303,

```
apply Box 2201
apply Box 2203, but use productPrice as the price for product iProdSup
for (iScenario = 0; iScenario < N_Sample; iScenario++)
   apply Boxes 2305 and 2307
```

3. In Box 2305, apply Boxes 2207 and 2209, except in Box 2209:

    - Limit the number of times Box 1551 is executed to *S/D_MaxRWItertions* times *mProd*
    - Limit the total time spent in Box 1551 to *S/D_MaxRWTime* seconds

4. In Box 2307, note produced quantity of product *iProdSup*.

5. In Box 2309, compute mean of noted produced quantity of Box 2307. This mean and *productPrice* determine a point of the supply schedule.

6. In Box 2311, write supply-schedule-data points to database. Update GUI database display.

    To also generate the average opportunity cost curve for *iProdSup*, the following is required:

    - At the start of Box 2301,

    ```
    set productPrice = 0
    set dSumBase = 0
    apply Boxes 2303, 2305, and 2307
    immediately after Box 2307, set dSumBase = dSumBase + |d|
    ```

    - At the start of Box 2303,

    ```
    dSumCase = 0
    qtSumCase = 0
    ```

    - At the end of Box 2307,

    ```
    set dSumCase = dSumCase + |d| - productPrice * (quantity of
       product iProdSup supplied)
    qtSumCase = qtSumCase + (quantity of product iProdSup
       supplied)
    ```

    - At the end of Box 2309, compute the average cost as being:

    ```
    (dSumBase - dSumCase)/ qtSumCase
    ```

Demand Simulation

The procedure to generate product demand schedules is shown in Figure 24. For expository purposes, the demand schedule generated is for a resource *iResDem* and will have quantities between *lowQuant* and *highQuant*. An offset, *offSetQuant*, needs to be a positive value. This procedure entails,

1. In Box 2401,

```
for (resourceQuant = lowQuant; resourceQuant < highQuant;
  resourceQuant = resourceQuant + increment)
    apply Boxes 2403 through 2407
```

2. In Box 2403, apply Boxes 2201 through 2211, except:
    - use *resourceQuant* minus *offSetQuant* as the quantity for *resQuant[iResDem]*
    - in Box 2211, only note the value of $|d|$
    - limit the number of times the loop defined by Box 1551 is executed to *S/D_MaxRWItertions* times *mProd*
    - limit the total time spent in Box 1551 to *S/D_MaxRWTime* seconds
3. In Box 2405, apply Box 2403, except:
    - use *resourceQuant* plus *offSetQuant* as the quantity for *resQuant*[*iResDem*]
4. In Box 2407, note the demand schedule point as having a price of:

    ((mean value of $|d|$ in Box 2405) -

    (mean value of $|d|$ in Box 2403))/*2* * *offSetQuant*

    and a quantity of *resourceQuant*.
5. In Box 2409, write demand-schedule-data points to database. Update GUI database display.

Use

This preferred embodiment envisions — almost requires — interaction with the present invention's user for two reasons:

- The best use of the present invention results from the interaction between the user and the invention. After reviewing simulation results, the user applies his or her knowledge to consider organizational resource, product, and marketing changes. Data changes are

made to reflect these considered changes, which are evaluated by the invention in subsequent simulations.

- The Resource-conduit process implicitly assumes that allocations can be shifted as Potential-demand changes. If such an assumption is not appropriate for the case at hand, then the user needs to experiment with different fixed-group allocations: *fixedAlloc* fields need to be set to "yes" and *fxAlQt* values specified; after a Base simulation, the resulting marginal value data suggests which fixed allocations the user should experimentally decrease and increase. The process of the user's setting fixed allocations and Base simulations being performed repeats until the user is satisfied with the resulting allocation.

The main purpose of the supply and demand schedules, and a major purpose of the marginal cost/value data, is to facilitate the user's considering and evaluating resource, product, and marketing changes. Many people responsible for allocating organizational resources — almost all MBAs — understand and know how to use supply schedules, demand schedules, and marginal costs/values.

Indirect Allocation

In order to apply indirect allocation, estimates of product demand distributions, resource requirements, and resource availabilities are used in a Base simulation with multiple scenarios. The resulting resource marginal values are then used as resource cost/price/value. If the value to be received is greater than or equal to the sum of component marginal costs, then the considered action should proceed.

For example, suppose that a Base simulation yielded the following marginal values for the following resources:

WI-cash	\$0.01
rxa1	\$5.00
rxb2	\$2.50

rxc3	\$3.00

And suppose that an opportunity (which may or may not have been anticipated in the Base simulation) becomes available and requires the following resource quantities:

rxa1	1
rxb2	3
rxc3	4

Further suppose that this opportunity requires \$30.00 for buyable resources and has a Fill-value of \$10.00. The opportunity cost of executing this opportunity is \$44.80 (5 * 1 + 3 * 2.50 + 4 * 3.00 - 10 + 30 * (1.00 + 0.01)). If the price to be received by the organization exceeds or equals \$44.80, it is in the organization's interest to execute the opportunity. Conversely, if the value to be received is less than \$44.80, it is not in the organization's interest to execute the opportunity.

The basis for this approach is two-fold. First, the Base simulation is a sampling of opportunities, optimal allocations, and marginal costs/values. Second, such marginal costs are opportunity marginal costs. Were a Base simulation rerun with a small resource quantity change, then the change in the object function value would be roughly equal to marginal cost times the resource change quantity.

Besides costing products, resource marginal values can be used to evaluate acquiring and divesting resources: if additional resource quantities become available at a price less than marginal value, it would be desirable to acquire the additional quantities; conversely, if an opportunity to sell a resource at a price greater than its marginal value manifests, it would be desirable to divest at least some of the resource.

Similarly to the way that an economy uses the free-market pricing mechanism to optimally allocate resources, an organization uses this invention's indirect pricing allocation method to

optimally allocate resources. The yielded marginal values determine where, when, and for what purpose a resource is used: a low value suggests a resource has a low value and consequently results in relatively casual use; conversely, a high marginal value suggests that a resource is precious and results in use only when the compensating payback is sufficiently high.

Indirect allocation is not as good as direct allocation, nor as good as comparing two base simulations — one with the resource quantities removed, the other with the resource quantities included. This is because approximations are being used to anticipate net results. However, because many organizations are in constant flux, there is never a moment when all allocations can definitively be optimized. For those organizations, and at such times, indirect allocation is the best alternative.

Conclusion, Ramifications, and Scope

Thus, as the reader who is familiar with the domain of the present invention can see, the invention leads to optimized or near-optimized allocations of organizational resources. With such optimization, organizations can better reach their goals.

While the above description contains many particulars, these should not be construed as limitations on the scope of the present invention, but rather, as an exemplification of one preferred embodiment thereof. As the reader who is skilled in the invention's domain will appreciate, the invention's description here is oriented towards facilitating ease of comprehension; such a reader will also appreciate that the invention's computational performance can easily be improved by applying both prior-art techniques and readily apparent improvements.

Many variations and add-ons to the preferred embodiment are possible. For example, without limitation:

1. When generating random *potentialDemand* values for each scenario, generate other random values for other data, such as for prices (elements in vector *c*), available resource quantities (vector *resQuant*), and product unit requirements (*reqQt* values placed in matrix ***a***). Such may require adjusting ***rcMat*** column group allocations so that they sum to *resQuant* (see Variation #22 for how this is done) and applying prior-art techniques to update linear programming memory.
2. When implementing the above Variation #1, generate correlated random numbers. For example, have the generated random prices be partly or completely correlated with the generated random *potentialDemand* values.
3. Allow buyable resources to be allocated to groups in the first *mProd* rows of ***rcMat***. This requires the introduction of a pseudo product that has an infinite Potential-demand, that has a price of one currency unit, and that has a unit fulfillment requirement of one WI-cash unit. (This pseudo product assures that the marginal return of WI-cash allocations to groups in the first *mProd* rows of ***rcMat*** is non-negative.) (See variation #20 on how to have multiple ***rcMat*** columns handle WI-cash.)
4. Allow multiple types or categories of WI-cash.
5. Generate *rowEffectivenesses* using other functional forms, besides the multiplicative form that is the focus of the present description. A function of the following form can be considered to generate *rowEffectivenesses*:

$$rowEffectiveness_i = AG_i(ef_{i,0}, ef_{i,1}, ef_{i,2}, ef_{i,3}, \ldots ef_{i,nRes-1})$$

where:

AG_i uses the effectivenesses of the elements in row i of ***rcMat*** to generate $rowEffectiveness_i$

$ef_{i,j}$ is the piecewise linear allocation-to-effectiveness function for the group having an element at $\boldsymbol{rcMat_{i,j}}$.

The AG function can in turn be considered to generate *rowEffectiveness* by using a hierarchy of cluster functions: Cluster functions pool group-element effectivenesses and possibly other cluster effectivenesses to generate cluster effectivenesses, which are in turn

used to generate other cluster effectivenesses, etc. — until a final cluster effectiveness, which is *rowEffectiveness*, is obtained.

What is desirable, but not necessary, is for AG_i to be directionally differentiable with respect to each $ef_{i,j}$, and associated allocation. When this is the case for a particular $ef_{i,j}$, then:

$$emcSub_{i,j} = ((maximized\ |c_i|\ value\ as\ generated\ in\ Box\ 1001) * bOrg_i/AG_i) * (\partial AG_i/\partial ef_{i,j}^{-}) * (\partial ef_{i,j}/\partial(allocation\ to\ group\ head)^{-})$$

$$emvAdd_{i,j} = ((minimized\ |c_i|\ value\ as\ generated\ in\ Box\ 1001) * bOrg_i/AG_i) * (\partial AG_i/\partial ef_{i,j}^{+}) * (\partial ef_{i,j}/\partial(allocation\ to\ group\ head)^{+})$$

If AG_i is not directionally differentiable, then $emcSub_{i,j}$ and $emvAdd_{i,j}$ can be determined by numerical methods or, alternatively, ignored by setting $emcSub_{i,j} = BIG\_M$ and $emvAdd_{i,j} = 0$

Group-head *maxSub* and *maxAdd* quantities need to be bounded by the maximum decrease and increase in the group allocation that can be made without changing $emcSub_{i,j}$ and $emvAdd_{i,j}$ respectively, holding $|c[i]|$ constant.

Irrespective of how $emcSub_{i,j}$ and $emvAdd_{i,j}$ are generated, the Axis-walk, Ridge-walk, and Lateral-walk processes can proceed as described. The Top-walk process could ignore uv-group pairs that are not part of a multiplicative cluster. Alternatively, Top-walk could perform special handling: the AG_i function needs to be algebraically converted to a function with a domain as the increment to the allocation of the group containing an element at $\boldsymbol{rcMat_{i,u}}$ and a range as the amount by which the allocation to the head of the group containing an element at $\boldsymbol{rcMat_{i,v}}$ can be reduced, while holding AG_i constant. This function defines the *TWvfu* routine that should be used for the $\boldsymbol{rcMat_{i,u}}$ and $\boldsymbol{rcMat_{i,v}}$ pair; the derivative defines the *TWvfuEpsilon* routine for the same pair. (*TWufv* and *TWufvEpsilon* are the inverse functions of *TWvfu* and *TWvfuEpsilon* respectively.)

Two particularly useful non-multiplicative cluster forms are, what are termed here, the sufficiency and complementary clusters. The sufficiency cluster has the following form:

$$clusterEffectiveness = 1 - \Pi(1 - ef_{ik})$$

where k iterates through all cluster group elements. This type of cluster is appropriate when more than one resource can accomplish the same fundamental conversion from Potential-demand to Realized-demand. For example, developing product awareness through advertising can be accomplished through television and radio. Once awareness (for a unit of Potential-demand) is obtain in one medium, awareness development activity in the other medium is not needed. This can be handled by separately estimating the allocation-to-effectiveness (awareness) function for one medium, assuming a zero effectiveness (awareness) for the other medium. Then the two media are aggregated using the sufficiency cluster.

The complementary cluster has the following form:

$$clusterEffectiveness = minimum\ effectiveness\ of\ group\ elements\ ef_{ik} \ldots$$

where k iterates through all cluster group elements. This type of cluster is appropriate when more than one resource must be used jointly to accomplish the same fundamental conversion from Potential-demand to Realized-demand. For example, design of a product could require that design and engineering resources work closely together and, as a consequence, could be of a nature that the allocation of each resource determines an upper bound on overall design effectiveness.

The allocation-to-effectiveness functions for the groups of the sufficiency, complementary, and other types of clusters can be determined in a manner similar to that described for the multiplicative cluster. In particular, by asking the following question: Presuming that a group's allocation is the only factor governing whether 0% to 100% of Potential-demand is converted to Realized-demand, how does the percentage vary as the allocation varies?

6. Use multiple parallel processors to share the processing burden.
7. Allow multiple users to simultaneously edit the database and run simulations.

8. Subtract committed resources and committed product quantities prior to the allocation process starting.

9. Eliminate the linear programming process when no fulfillment allocations are made or needed. This can be accomplished by using the above Resource-conduit process without linear programming processing and:

   - always using the negative value of *c[m+i]* set in Box 2203 for the value of *c[i]* used in Box 1001
   - presuming that if *b* vector values were to be generated, they would always be positive.
   - calculating |*d*| by summing each product's working price (*c[m+i]*) times the *bOrg[i]* quantity.

10. Ignore optimizations and determine resource marginal costs/values and product marginal costs for an allocation plan not formulated by this invention. This entails fixing allocations, including the linear programming allocations, to reflect the allocation plan and then computing *gmcSub*, *gmvAdd*, *twmcSub*, etc.

11. Incorporate prior-art linear programming techniques, such as (without limitation), sparse matrix, ellipsoid, and integer (programming) techniques.

12. Correct for accumulated rounding errors: Reapportion *resQuant* and regenerate Resource-conduit data as follows:

```
for (j = 0; j < nRes; j++)
   set sum = sum of allocations in column j of rcMat
   for (each group head in column)
      set allocation = (allocation/sum) * resQuant[j]
   regenerate group effectiveness
regenerate bOrg
```

   Then apply prior-art linear-programming techniques to re-invert ***B*** and freshly generate the simplex tableau.

13. Allow the user to specify a group's allocation-to-effectiveness function as a formula. This would require:

   - immediately each time after the group's allocation is changed,

```
set maxSub and maxAdd such that:
  • both are non-negative
```

```
• group allocation - maxSub is in the domain of the specified
  formula
• group allocation + maxAdd is in the domain of the specified
  formula
(The smaller the values for maxSub and maxAdd, the more numerically
accurate the final allocation, but the more processing time
required.)
set atoeFnPt[0].allocation = allocation - maxSub
set atoeFnPt[0].effectiveness = functional value of (allocation -
  maxSub)
set atoeFnPt[1].allocation = allocation
set atoeFnPt[1].effectiveness = functional value of (allocation)
set atoeFnPt[2].allocation = allocation + maxAdd
set atoeFnPt[2].effectiveness = functional value of (allocation +
  maxAdd)
set dedaSub = slope of line between points atoeFnPt[0] and
  atoeFnPt[1]
set dedaAdd = slope of line between points atoeFnPt[1] and
  atoeFnPt[2]
set ir = 1
```

- as in the Top-walk and Lateral-walk processes, include in the Axis-walk process block clearing, pair blocking, shift evaluation, and shift reversal.

14. Allow nonlinear fulfillment allocations. To do this, during initialization, below the first *mProd* rows of matrixes ***rcMat*** and ***a***, insert an empty row. Afterwards, place one or more group heads in the inserted row. In matrix ***a***, place a 1.0 in the inserted row and column corresponding to the product for which the nonlinear fulfillment allocation is to be allowed. Also place a 1.0 in the corresponding element of *potentialDemand*. Analogously to before, allocations to the group(s) of the inserted row determine group effectiveness, which in turn determines a *rowEffectiveness* value, which in turn determines a *bOrg* value, which in turn sets an upper bound to the number of units that can be made, given the resources allocated to the group(s) of the inserted row.

When increasing the *rowEffectiveness* for the product with nonlinear fulfillment allocations in the Ridge-walk process, allocations need to be shifted into and out of the groups of the inserted row. This is the same as what was done in Box 1507 vis-à-vis individual Direct-put groups. If there is more than one group element in the inserted row and the inserted row has a tighter bound (i.e. *bOrg[inserted row]* < *bOrg[rwiRow]*), then a separate,

independent, parallel Ridge-walk process needs to increase the allocations to the groups of the inserted row until the bound is relieved.

15. Capitalize on congruent Top-walk cycles. When generating *twmcSub* values, cycles can develop where each group in a cycle alternatively entertains compensatory allocations from other groups in the cycle and the *twmcSub* values decrease to 0. Performance can be improved by testing for such cycles, and upon discovery, directly setting all cycle *twmcSub* values to 0.

    Similarly, a Top-walk chain can end in a cycle where costless allocation-shifting-out of a cycle can occur because, in essence, an arbitrage opportunity is being exploited. When this occurs, it is preferable to extract what can be extracted from the cycle, shift the extract through the remainder of the chain, update *bOrg*, make feasible the linear programming problem, and avoid matrix multiplication to determine *twQuant*.
16. When doing the Top-walk process, generate a *twmvAdd* (marginal value add) value, in place of, or in addition to, *twmcSub*. The Top-walk process as described has a subtraction orientation: the allocation in one group decreases, a compensatory allocation increase is made, which in turn requires another allocation decrease, etc. The orientation could, just as well, be reversed: the allocation in one group increases, which makes possible the allocation decrease in another group, which in turn triggers another possible allocation increase, etc.
17. Use *gmvAdd* (rather than, or in addition to, *gmcSub*) when generating for display and database-storage resource marginal values, distribution marginal values, and/or product marginal costs.
18. Include other data in the database, in particular, data that would facilitate comparison between marginal costs and open-market resource prices.
19. During the initialization process, if two or more resources are perfect complements, meaning they are always used jointly in the same proportions, then merge the complementary resources into a single combined resource.

20. Allow a single resource to span multiple ***rcMat*** matrix columns. Processing can proceed as described above for the preferred embodiment, except that the multiple columns need to be handled as if they were a single column when searching for the minimum *gmcSub* and *twmcSub*. This would allow the allocation of a resource type which, in effect, is transformed or specialized upon allocation. For instance, if the resource were cash, then implicitly a conversion to, for example, engineering or design resources might be taking place upon allocation.
21. Allow allocations to genuinely span multiple time periods. Initially load data that is specific to each time period into its own version of the memory shown in Figures 4 and 5. When doing this loading, vector *c* values should be appropriately discounted. Then merge the time-period formulations into a master version of the memory shown in Figures 4 and 5. Initially, this master version has no inter-period ties: the allocations of one period are independent of the allocation of another period, and the layout of utilized memory is highly "rectangular."

    Then use standard linear programming techniques to perform inter-period ties, to, for instance, handle WI-cash being increased, decreased, and passed to subsequent periods. WI-cash payouts and receipts should be time-phased so that WI-cash for each period is accurately determined and available for subsequent periods' buyable resources. Payouts and receipts that belong to beyond the last time period should be consolidated into the last time period (when *WI-cash Type = Fold-in*) or ignored (when *WI-cash Type = Spread-out*).

    Where appropriate, consolidate and duplicate ***rcMat*** columns and group elements; where appropriate, introduce AG clustering (see Variation #5). For instance, an allocation to a design group in one period might complement a design group of another period. In this case, duplicate a group element from the earlier time period into the latter period. Then add a sufficiency cluster in the latter time period to aggregate the design effectivenesses of both the earlier and the latter groups. (This sufficiency cluster might want to discount the earlier period's effectiveness.)

Each scenario would comprise several sequential time periods. Potential-demands for all time periods would be generated simultaneously, and the allocation process would simultaneously apply to all periods. As before, a simulation could entail one or more scenarios, and could have unfulfilled Potential-demand being passed on to subsequent time periods and scenarios. For instance, unfulfilled Potential-demand of period 2 scenario 7 would be passed onto period 3 of scenario 8.

22. Reuse Base scenario solutions for Supply and Demand and subsequent Base simulations. When doing a first Base simulation, save the linear programming and Resource-conduit solution after each scenario. When doing a subsequent simulation, prior to each scenario, restore the saved solution and use it as a starting point. If in an ***rcMat*** column the sum of group allocations is greater than *resQuant*, then subtract group allocations from the groups having the smallest *gmcSubs* until the sum of group allocations equals *resQuant*. Conversely, if the sum is less than *resQuant*, add to groups with the largest *gmvAdd*. Use prior-art techniques to make the linear programming solution both feasible and optimized. Afterwards, optimize the totality, as described.
23. Relax the thoroughness of the optimization in order to reduce the required number of CPU cycles. For instance, without limitation, skip any combination of the following:
    - Box 705 (Axis-walk)
    - Box 707 (Top-walk)
    - Box 709 (Lateral-walk)
    - Box 711 (Ridge-walk)
    - Boxes 1305 through 1331, inclusive
    - Boxes 1307 through 1317, inclusive
    - Boxes 1329 through 1331, inclusive
    - Boxes 1611 through 1627, inclusive
    - Box 1551 for some or all rows of ***rcMat***
    - Box 1511

- The Top-walk and/or the Lateral-walk processes in Box 1511.
- The Top-walk portion of Box 1407, i.e., attempt only an Axis-walk iteration – which is implicitly included in Top-walk.

24. Include capability for the user to integrate, i.e. find the area beneath, the generated supply and demands schedules.
25. Use user-friendly column titles. Figures 18 - 21 are oriented towards the technical discussion. The titles listed below are oriented towards the user and are the preferred titles for actual use. Specifically,
    - For the Distributions Window:

Row	Column	Title
0	1	Name
0	2	Type
0	7	Distribution

- For the Resources Window:

Row	Column	Title
0	1	Name
0	4	Quantity
0	14	Demand
1	2	Group Name
1	3	Fixed Alloc
1	4	Allocation
1	9	Effectiveness

- For the Products Window:

Row	Column	Title

0	1	Name
0	2	Price
0	3	Fill-value
0	4	Dist Pot
0	5	DP %
0	14	Supply
1	2	Resource
1	3	Quantity
1	4	Periods to Cash

26. Use several different values for *facReduce* in the Lateral-walk process.

27. Use a modified Lateral-walk process. This process could be used in addition to the normal Lateral-walk process and entails:

```
for (i = 0;i < mProd; i++)
  if exist a group element in row i of rcMat such that in its group
    head there exists an irx such that the slope of line segment irx is
    less than the slope of line segment irx+1 (as defined in Box 909)
        Apply Box 709, except replace Box 1405 with:
          set potentialDemand[i] = potentialDemand[i] * facReduce
          apply Box 911
          set b = B * bOrg
          apply box 1337
```

Ideally, different values between 0 and 1 should be used for facReduce in this modified version of Lateral-walk. This modified version might be termed Explode-walk.

28. Include fixbuy as a hybrid between the fixed and buyable resource types. An example of such a resource would be office space obtained under a long term contract. It entails a fixed periodic payment and its availability is fixed. Processing would proceed as follows: the fixed periodic payment would be subtracted from WI-cash as the *resQuant* array is initially populated; in all other regards, it would be handled as a fixed resource.

29. Experiment with ***rcMat*** initializations and Monte Carlo search. Specifically, repeat the following several times (each time constituting an instance): initially randomly allocate *resQuant* to groups (instead of using the proportional method of Box 701), generate effectivenesses, generate *rowEffectiveness*, generate *bOrg*, ... , and compute $|d|$. Next, randomly do or not do each of the following any number of times and in any order:
    a) Apply some or all of the Walk processes to some or all of the instances.
    b) Discard instances with low $|d|$.
    c) Within individual instances, randomly shift allocations between groups of the same ***rcMat*** column .

    Then accept, as a final allocation, the instance that yields the highest $|d|$. This Variation #29 might be called a Rand-mode process.

30. Enhance Variation #29 by combining allocations from different instances to form additional instances. For instance, suppose there are *nGroup* groups and currently *nStance* instances. Create an additional instance by:

```
for (i = 0; i < nGroup; i++)
   Randomly select an instance that yields one of the higher |d|s.
   Set group i allocation = allocation of group i in randomly selected
      instance.
Randomly increase or decrease group allocations so that for each column
   of rcMat, the sum of group allocations equals resQuant.
```

    This variation #30 is arguably a genetic algorithm, and might be called a Genetic-mode process.

The following is the software listing for carrying out the invention.

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]colrowid.h[~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~
#ifndef ColRowId_H
enum COLNAME {cnNone, cnDName, cnDType, cnDMV, cnDMean, cnDDist,
cnRName,
cnRUnit, cnRAvailability, cnRQuanity, cnRWTMD, cnRPayPrice,
cnRMV,
cnRMeanUse, cnRDemand, cnRGName, cnRGDedicate, cnRGAllot,
cnRGMV,
cnRGMeanUse, cnRGEffectiveness, cnRGDemand, cnPName, cnPPrice,
cnPFillValue, cnPDist, cnPDistPC, cnPCOver, cnPMC,
cnPMeanSupply,
cnPMeanDemand, cnPSupply, cnPResource, cnPQuantity, cnPDtoCash,
cnFTitle,
cnFNext, cnFPrev0, cnFCur, slNone, slDistribution,
slAvailability,
slNoYes, slYesNo, slResourceName, slCType, slAllocType,
slMaxType,
slMCDisplayType}; extern int colDName; extern int colDType;
extern int colDMV; extern int colDMean; extern int colDDist;
extern int colRName; extern int colRUnit; extern int
colRAvailability;
extern int colRQuanity; extern int colRWTMD; extern int
colRPayPrice;
extern int colRMV; extern int colRMeanUse; extern int
colRDemand;
extern int colRGName; extern int colRGDedicate; extern int
colRGAllot;
extern int colRGMV; extern int colRGMeanUse;
extern int colRGEffectiveness; extern int colPName; extern int
colPPrice;
extern int colPFillValue; extern int colPDist; extern int
colPDistPC;
extern int colPCOver; extern int colPMC; extern int
colPMeanSupply;
extern int colPMeanDemand; extern int colPSupply; extern int
colPResource;
extern int colPQuantity; extern int colPDtoCash; extern int
colFNext;
extern int colFPrev0; extern int colFPrevn; extern int colFCur;
extern int iCashRow; enum ROWNAME {rowFBlank = -1, rowFIPS = 1,
rowFIPSse = 2, rowFDCash = 4, rowFDCashse = 5, rowFCash = 7,
rowFCashMV = 8, rowFCashMVse = 9, rowFSFValue = 11,
rowFSFValuese = 12,
rowFSWTDM = 14, rowFSWTDMse = 15, rowFPara = 17, rowFAType = 18,
rowFMax = 19, rowFCType = 20, rowFRSeed = 21, rowFNSample = 22,
rowFMCDisplay = 23, rowFRWBaseIter = 25, rowFRWBaseTime = 26,
rowFRWCaseIter = 28, rowFRWCaseTime = 29};
#define fcurCol 2
#define ColRowId_H
```

```
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]colrowid.cpp[~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~
#include "stdafx.h"
#include "ColRowId.h"
int colDName = -1; int colDType = -1; int colDMV = -1; int
colDMean = -1;
int colDDist = -1; int colRName = -1; int colRUnit = -1;
int colRAvailability = -1; int colRQuanity = -1; int colRWTMD =
-1;
int colRPayPrice = -1; int colRMV = -1; int colRMeanUse = -1;
int colRDemand = -1; int colRGName = -1; int colRGDedicate = -1;
int colRGAllot = -1; int colRGMV = -1; int colRGMeanUse = -1;
int colRGEffectiveness = -1; int colPName = -1; int colPPrice =
-1;
int colPFillValue = -1; int colPDist = -1; int colPDistPC = -1;
int colPCOver = -1; int colPMC = -1; int colPMeanSupply = -1;
int colPMeanDemand = -1; int colPSupply = -1; int colPResource =
-1;
int colPQuantity = -1; int colPDtoCash = -1; int colFNext = -1;
int colFPrev0 = -1; int colFPrevn = -1; int colFCur = -1;
int iCashRow = -1;
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]cort.h[~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
#include "Jtools.h"
#ifndef Cort_H
extern prec cthQuant[CORTMAX_N]; extern prec cthMC [CORTMAX_N];
extern   prec   ctvQuant[CORTMAX_M1];   extern   prec   ctvMVs
[CORTMAX_M1];
extern    prec    ctvMVa    [CORTMAX_M1];    extern    BOOL
ctvMVcur[CORTMAX_M1];
extern prec ctProfit; void CortPrep(int setm, int setn, int
setmProd);
void    CortSetInitial();    void    CortUnPrep();    void
CTLoadaElement(int i,
int jr, prec value); prec CTGetOrgaElement(int ibResource, int
jProd);
void CTFactorInb(prec* pData); void CTFactorInb(NEXTs& ns);
void CTLoadb(prec* pData); void CTFactorInc(prec* pData);
void CTLoadc(prec* pData); void CTSwampPriceInit();
void   CTSwampPrice(int   iProd,   prec   factorUp);   void
CTSwampPriceReverse();
BOOL CTFinLoad(BOOL scale); void CTMakeLoadCreatedSpace();
BOOL CTScaleCol(int coljr); BOOL CTScaleRow(int row);
prec CTGetBound(int irow, int jProd); prec GenCellR(int i, int
j);
void GenRow(int i, int jbut); void GenCol(int j, int ibut); void
Genb();
void   Genc();   void   Gend();   void   FlushbOrgPrep();   void
FlushbOrgNote(int i);
void    FlushbOrgFin();    int    CTInvertMatrix();    void
CTRoundAdjustment();
```

```
int CTMaximize(); int CTMakeFeasible(); void CTIncb(prec facIn,
NEXTs& bSupl, prec& facOut, int& rtCond); int CTReMax();
void Pivot(BOOL bUpdate, BOOL BrUpdate=FALSE, prec*
pbpbaseb=NULL,
prec* pbpcurb=NULL); void Pivot(int i, int j, BOOL bUpdate);
int PivotRowOut(int outRow, BOOL bUpdate, prec* pbpcurb =NULL);
void FinMaxMF(BOOL checknegb =TRUE); void NextjInsert(int j);
void NextjDelete(int j); void CyclRepeatPrep();
BOOL CyclRepeating(int index); void CTbPlayBbyVec(prec* inb,
prec* outb);
void CTbPlayBbyVec(NEXTs& inbNext, prec* inb, prec* outb);
void CTbPlayBbyVec(NEXTs& inbNext, prec* inb, int irow, prec&
outbele);
int CTbPlayLoadbbOrg(prec* inb, prec* inbOrg, BOOL negb);
void CTbPlayPrepMaxNegb(); void CTbPlayMaxNegb(prec* pbpbaseb,
prec* pbpcurb, int& bpcurmini, prec bpTipOver);
BOOL CTShiftNeeded(int jShift); void CTShiftBOrgIn(int iShift,
prec facIn,
prec& facOut); void CTClearGetvh(); prec CTGetProfit();
void CTNoteDolProfitAlso(); void CTSetProfit(prec profit);
prec CTGethQuant(int jr); prec CTGetRawProdQuant(int jProd);
void CTGethData(int jr, int nOrder, int iStart=-1); int
CTGetNumOb();
int CTIsZerob(int i_brow); BOOL CTv2MV(int i_brow);
void CTGetvQData(int i); void CTGetvDataFin(int i); void
CTGetvData(int i)
; prec CTGetb(int i); prec CTGetbWT(int i); prec
CTGetbOrgBak(int i);
prec CTGetbOrg(int i); prec CTGetbOrgWT(int i); prec CTGetc(int
j);
prec CTGetcOrg(int jr); void ClearvMV(); long double
CTGetPivotCount();
prec MincElement(int j); prec MaxcElement(int j);
void GetCortDim(int& getm, int& getn, int& getmProd);
class CTstor : public CObject {DECLARE_SERIAL(CTstor); private:
int level;
prec tableuOrg[CORTMAX_M1][CORTMAX_N+1]; long double
hScale[CORTMAX_MN+1],
vScale[CORTMAX_M1]; prec cOrg[CORTMAX_N]; prec bOrg[CORTMAX_M1];
prec cFactorIn[CORTMAX_MN], bFactorIn [CORTMAX_M1], dFactorIn;
prec B[CORTMAX_M1][CORTMAX_MN]; int jstartL, jstartR; prec
b[CORTMAX_M1];
prec cR[CORTMAX_MN]; int jnext[CORTMAX_MN+1]; int u[CORTMAX_M1];
int uref[CORTMAX_MN]; NEXTs nextOb; BOOL dgood; prec ctvMVs
[CORTMAX_M1];
prec ctvMVa [CORTMAX_M1]; BOOL ctvMVcur[CORTMAX_M1]; public:
CTstor();
void Out(int setlevel); void OutvMV(); void In(); void
InvMV();};
class CTstorp : public CObject {DECLARE_SERIAL(CTstorp);
private: CTstor* pCTstor; public: CTstorp(); ~CTstorp();
```

```
void Out(int setlevel); void OutvMV(); void In(); void
InvMV();};
#define Cort_H
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]cort.cpp[~~~~~~~~~~~~~~~~~~~~~~
~~~~~~
#include "stdafx.h"
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "jtools.h"
#include "cort.h"
#include "rcdt.h"
#include <float.h>
#include <math.h>
#include <limits.h>
#include "jtools.h"
#include "cort.h"
prec DolGetProfit();
#define rowL B[irow]
#define cL B[m]
#define GenCellL(i, j) B[i][j]
#define GenCell(i, j) (j<m) ? GenCellL(i,j) : GenCellR(i,j)
#define jLOOPL for(j=jstartL;j<m; j=jnext[j])
#define jLOOPR for(j=jstartR;j<mn;j=jnext[j])
long double tm[CORTMAX_M1][2 * CORTMAX_M1]; prec
tableuOrg[CORTMAX_M1]
[CORTMAX_N+1]; long double hScale[CORTMAX_MN+1],
vScale[CORTMAX_M1];
NEXTc aCol[CORTMAX_N] = {NULL}; NEXTc aRow[CORTMAX_M1] = {NULL};
prec cOrg[CORTMAX_N]; prec bOrg[CORTMAX_M1] = {0};
prec bOrgBak[CORTMAX_M1]; prec cFactorIn[CORTMAX_MN],
bFactorIn [CORTMAX_M1], dFactorIn; prec
B[CORTMAX_M1][CORTMAX_MN]; int m,
n, mn, m1, n1, jstartL, jstartR, irow, jcol; prec b[CORTMAX_M1];
prec rowR[CORTMAX_MN], cR[CORTMAX_MN]; prec col[CORTMAX_M1];
int jnext[CORTMAX_MN+1]; int u[CORTMAX_M1]; int
uref[CORTMAX_MN];
int ctmProd; NEXTs nextTemp; NEXTs next0b; BOOL dgood; BOOL
dolProfitAlso;
BOOL flushbOrgpend; prec flushbOrgc[CORTMAX_N];
prec flushbOrgTolerance[CORTMAX_M1]; BOOL reMaxNecessary =
FALSE;
prec cyclRepeatbm; int cyclRepeatIndex; int cyclRepeatCount;
int cyclRepeatjnext[CORTMAX_MN]; long double pivotCount = 0;
long double loadcPivotCount = -1;
#define ZEROCUTCPV 0
#define ZEROCUTSTO 0
#define ZEROCUTNEXT0b TOLERANCE
prec cthQuant[CORTMAX_N]; prec cthMC [CORTMAX_N]; prec
ctvQuant[CORTMAX_M1]
; prec ctvMVs [CORTMAX_M1]; prec ctvMVa [CORTMAX_M1];
```

```
BOOL ctvMVcur[CORTMAX_M1]; prec ctProfit; BOOL vMVClear;
void CortPrep(int setm, int setn, int setmProd) {CortUnPrep(); m
= setm;
n = setn; ctmProd = setmProd; mn = m + n; m1 = m + 1; n1 = n +
1;
CortSetInitial(); ZEROOUT2D(tableuOrg, (CORTMAX_M1), (CORTMAX_N
+1));
dgood = TRUE; NextPrep(next0b, m); NextPrep(nextTemp,mn);
SPREAD(vScale ,
m1 , 1); SPREAD(hScale ,mn+1, 1); SPREAD(bFactorIn,m , 1);
SPREAD(cFactorIn,mn , 1); dolProfitAlso = FALSE;} void
CortSetInitial()
{int i,j; ZEROOUT2D(B, m1, m); for(i=0;i<m;i++) {B[i][i] = 1.0;
u[i] = i;
uref[i] = i;} jstartL = m; jstartR = m; ZEROOUT(jnext[0],m);
for(j=m;j<mn;
j++) jnext[j] = j + 1; jnext[mn] = mn;} void CortUnPrep() {int
i, jr;
for(jr=0; jr<CORTMAX_N; jr++) NextClear(aCol[jr]); for(i=0; i
<CORTMAX_M1;
i++) NextClear(aRow[i]);} void CTLoadaElement(int i, int jr,
prec value)
{tableuOrg[i][jr] = value;} prec CTGetOrgaElement(int
ibResource,
int jProd) {return tableuOrg[ibResource][jProd];}
void CTFactorInb(prec* pData) {for(int i=0; i<m; i++) pData[i]
*= bFactorIn[i];} void CTFactorInb(NEXTs& ns) {int i; iLOOPs(ns)
ns.val[i]
*= bFactorIn[i];} void CTLoadb(prec* pData) {ARRAYCOPY(bOrg[0],
bOrgBak[0]
, m); ARRAYCOPY(pData[0], bOrg[0], m); if(jstartL == m)
{ARRAYCOPY(bOrg[0],
b[0], m) b[m] = 0;} else {Genb(); dgood = FALSE;}}
void CTFactorInc(prec* pData) {for(int jr=0; jr<n; jr++)
pData[jr]
*= cFactorIn[m+jr];} void CTLoadc(prec* pData) {if(jstartL == m)
{ARRAYCOPY(pData[0], cR[m], n)} else {int i, j;
for(j=m;j<mn;j++)
if(jnext[j]) cR[j] = pData[j-m] + (cR[j] - cOrg[j-m]);
else {prec factor = pData[j-m] - cOrg[j-m]; if(factor)
{GenRow(uref[j],mn)
; if(TRUE) {int j; jLOOPL cL[j] -=rowL[j] * factor; jLOOPR cR[j]
-=rowR[j]
* factor;}}} iLOOPs(next0b) if(b[i] <= 0) reMaxNecessary = TRUE;
dgood = FALSE; vMVClear = FALSE; ClearvMV();}
ARRAYCOPY(pData[0], cOrg[0],
n);} prec spHold[CORTMAX_N], spWork[CORTMAX_N],
swampPrice[CORTMAX_N];
void CTSwampPriceInit() {static prec aa[CORTMAX_M1][CORTMAX_N];
int i,j;
int iProd; for(i=0;i<m;i++) for(j=0;j<n;j++) aa[i][j] =
GenCellR(i,m+j);
```

```
ZEROOUT(swampPrice[0],n);    for(iProd=0;    iProd<n;    iProd++)
for(j=0;j<n;j++)
if(iProd != j) for(i=0;i<m;i++) if(aa[i][iProd] && aa[i][j])
{swampPrice[iProd] = __max((aa[i][iProd]/aa[i][j]) * cR[m+j],
swampPrice[iProd]);}      for(iProd=0;      iProd<n;      iProd++)
swampPrice[iProd]
*= (1.1 * (n - 1));} void CTSwampPrice(int iProd, prec factorUp)
{ARRAYCOPY(cOrg[0], spHold[0], n); ARRAYCOPY(cOrg[0], spWork[0],
n);
spWork[iProd] = swampPrice[iProd] * factorUp; CTLoadc(spWork);
CTMaximize();}   void   CTSwampPriceReverse()   {CTLoadc(spHold);
CTMaximize();}
BOOL CTFinLoad(BOOL scale) {int i,j,jr; BOOL withinTolerance =
TRUE;
SPREAD(vScale,m1, 1); SPREAD(hScale,mn+1, 1); for(i=0;i<m;i++)
tableuOrg[i][n] = bOrg[i]; for(jr=0;jr<n;jr++) tableuOrg[m][jr]
= cOrg[jr]
; if(scale) {int k; CTScaleCol(n); CTScaleRow(m); for(k=0;
k<25  &&  !withinTolerance;  k++)  {withinTolerance  =  TRUE;
for(jr=0;jr<n1;
jr++)    if(!CTScaleCol(jr))    withinTolerance    =    FALSE;
for(i=0;i<m1;i++)
if(!CTScaleRow(i)) withinTolerance = FALSE;}} for(i=0;i<m;i++)
{bFactorIn[i] = (prec) (vScale[i] * hScale[mn]); bOrgBak[i] =
bigM; b[i]
=  bOrg[i]  =  bOrg[i]  *  bFactorIn[i];  hScale[i]  =  (prec)
(1.0/vScale[i]);
flushbOrgTolerance[i]    =    0.007f    *    bFactorIn[i];}
for(j=0;j<mn;j++)
cFactorIn[j]    =    (prec)    (vScale[m]    *    hScale[j]);
for(jr=0;jr<n;jr++)
{cOrg[jr] *=cFactorIn[m+jr]; cR[m+jr] = cOrg[jr];} dFactorIn =
(prec)
(vScale[m] * hScale[mn]); CTMakeLoadCreatedSpace(); vMVClear =
FALSE;
ClearvMV(); CTSwampPriceInit(); return withinTolerance;}
void CTMakeLoadCreatedSpace() {int i, jr;
static      prec       rightaOrgWt[CORTMAX_M1][CORTMAX_N+1];
for(i=0;i<m;i++)
for(jr=0;   jr<n;   jr++)   rightaOrgWt[i][jr]   =   (prec)
(tableuOrg[i][jr]
*  (vScale[i]  *  hScale[m+jr]));  for(jr=0;  jr<n;  jr++)
{NextClear(nextTemp);
NextClear(aCol[jr]); for(i=0;i<m;i++) if(tableuOrg[i][jr])
NextInsert(nextTemp,      i,      rightaOrgWt[i][jr]);
NextCompress(nextTemp,TRUE,
aCol[jr]);}      for(i=0;i<m;i++)      {NextClear(nextTemp);
NextClear(aRow[i]);
for(jr=0; jr<n; jr++) if(tableuOrg[i][jr]) NextInsert(nextTemp,
jr,
rightaOrgWt[i][jr]); NextCompress(nextTemp,TRUE,aRow[i]);}}
BOOL CTScaleCol(int coljr) {int i; long double lo = prec_MAX;
```

```
long double hi = -prec_MAX; long double temp; for(i=0;i<m1;i++)
if(tableuOrg[i][coljr])  {temp  =  fabsl(tableuOrg[i][coljr])  *
vScale[i];
if(temp < lo) lo = temp; if(hi < temp) hi = temp;} if(hi == lo)
{hScale[m+coljr] = (prec) (1.0/hi); return TRUE;} else if(lo ==
prec_MAX)
return  TRUE;  else  {hScale[m+coljr]  =  (prec)  sqrtl(1.0/(lo  *
hi));
hi *= hScale[m+coljr]; lo *= hScale[m+coljr]; lo = lo/hi;
if(lo < TOLERANCE) return FALSE; else return TRUE;}}
BOOL CTScaleRow(int row) {int jr; long double lo = prec_MAX;
long    double    hi    =    -prec_MAX;    long    double    temp;
for(jr=0;jr<n1;jr++)
if(tableuOrg[row][jr])   {temp   =   fabsl(tableuOrg[row][jr])   *
hScale[m+jr];
if(temp < lo) lo = temp; if(hi < temp) hi = temp;} if(hi == lo)
{vScale[row]  =  (prec)  (1.0/hi);  return  TRUE;}  else  if(lo  ==
prec_MAX)
return TRUE; else {vScale[row] = (prec) sqrtl(1.0/(lo * hi));
hi  *=  vScale[row];  lo  *=  vScale[row];  lo  =  lo/hi;  if(lo  <
TOLERANCE)
return FALSE; else return TRUE;}} prec CTGetBound(int irow, int
jProd)
{if(TOLERANCE < tableuOrg[irow][jProd]) return (bOrg[irow]
/  (bFactorIn[irow]  *  tableuOrg[irow][jProd]));  else  return
bigM;}
prec  GenCellR(int  i,  int  j)  {int  k;  prec  element=0;
for(k=0;k<aCol[j-m]
.mele;k++) element += B[i][aCol[j-m].use[k]] * aCol[j-m].val[k];
return element;} void GenRow(int i, int jbut) {int j; irow = i;
if(jbut<m)
jbut  =  mn;  ZEROOUT(rowR[m],  n);  j  =  jstartR;  while(j<jbut)
{rowR[j]
= GenCellR(irow,j);  j  =  jnext[j];}  if(j==jbut)  j  =  jnext[j];
while(j<mn)
{rowR[j]  =  GenCellR(irow,j);  j  =  jnext[j];}  rowR[u[irow]]  =
1.0;}
void  GenCol(int  j,  int  ibut)  {int  i;  jcol  =  j;  if(j<m)
for(i=0;i<m1;i++)
col[i]  =  GenCellL(i,j);  else  {for(i=0;i<ibut;i++)  col[i]  =
GenCellR(i,j);
for(i=ibut+1;i<m;i++) col[i] = GenCellR(i,j); col[m] = cR[j];}}
void  Genb()  {int  i,j;  for(i=0;i<m;i++)  {if(u[i]  <  m)  b[i]  =
bOrg[u[i]];
else b[i] = 0; jLOOPL b[i] += B[i][j]*bOrg[j];} dgood = FALSE;}
void  Genc()  {int  j;  for(j=m;j<mn;j++)  if(jnext[j])  cR[j]  =
GenCellR(m,j)
+ cOrg[j-m]; else cR[j] = 0.0; dgood = FALSE;} void Gend() {int
i; b[m]
= 0.0; for(i=0;i<m;i++) if(m<=u[i]) b[m] -= b[i] * cOrg[u[i]-m];
dgood       =       TRUE;}       void       FlushbOrgPrep()
{ARRAYCOPY(cOrg[0],flushbOrgc[0],n);
```

```
flushbOrgpend = FALSE;} void FlushbOrgNote(int i) {if(bOrg[i]
<= flushbOrgTolerance[i] && bOrg[i] < bOrgBak[i] &&
ZeroPress(b[i]))
{int jr, k; LOOPc(aRow[i],k) if(!jnext[m+k]) {LOOPc(aRow[i],jr)
{flushbOrgc[jr] = -1; flushbOrgpend = TRUE;} break;}}} void
FlushbOrgFin()
{if(flushbOrgpend) {int ii; prec chold[CORTMAX_N];
ARRAYCOPY(cOrg[0],
chold[0],n); CTLoadc(flushbOrgc); CTMaximize();
CTInvertMatrix(); for(ii=0;
ii<m;ii++) if(!ZeroPress(bOrg[ii])) {if(jnext[ii]) {int i, mini;
prec minv;
GenCol(ii,m); mini = -1; minv = prec_MAX; for(i=0;i<m;i++)
if(0 < ZeroPress(col[i])) if(b[i]/col[i] < minv) {minv =
b[i]/col[i];
mini = i;} GenRow(mini,jcol); rowR[jcol] = col[irow];
Pivot(TRUE);}
b[uref[ii]] = 0;} CTLoadc(chold); CTMaximize();}} int
CTInvertMatrix()
{int i,j, d, maxi, m2=m+m; long double factor, maxv;
ZEROOUT2D(tm,m1,m2);
for(i=0; i<m; i++) tm[i][m+i] = 1.0; for(j=0;j<m;j++)
if(!jnext[j]) tm[j]
[uref[j]] = 1.0; for(j=m;j<mn;j++) if(!jnext[j]) {iLOOPc(aCol[j-
m]) tm[i]
[uref[j]] = aCol[j-m].val[aCol[j-m].cur]; tm[m][uref[j]] =
cOrg[j-m];}
for(d=0;d<m;d++) {maxi = d; maxv = fabsl(tm[d][d]);
for(i=d+1;i<m;i++)
if(maxv < fabsl(tm[i][d])) {maxv = fabsl(tm[i][d]); maxi = i;}
if(maxi != d && tm[maxi][d]) if(0 <= (tm[d][d]/tm[maxi][d]))
for(j=d;j<m2;
j++) tm[d][j] += tm[maxi][j]; else for(j=d;j<m2;j++) tm[d][j] -=
tm[maxi]
[j]; if(!(tm[d][d])) return (1); for(i=d+1;i<m1;i++)
if(tm[i][d])
{factor = -tm[i][d]/tm[d][d]; for(j=d+1;j<m2;j++) tm[i][j] +=
tm[d][j]
* factor;}} for(d=m-1;0<=d;d--) {for(i=0;i<d;i++) if(tm[i][d])
{factor = -tm[i][d]/tm[d][d]; for(j=m;j<m2;j++) tm[i][j] +=
tm[d][j]
* factor;} for(j=m;j<m2;j++) tm[d][j] /= tm[d][d];}
for(i=0;i<m1;i++)
for(j=0;j<m;j++) B[i][j] = (prec) tm[i][m+j]; Genb(); Genc();
dgood = FALSE; vMVClear = FALSE; return (0);} void
CTRoundAdjustment()
{CTInvertMatrix(); CTMakeFeasible(); CTMaximize(); vMVClear =
FALSE;
ClearvMV();} int CTMaximize() {int i,j,maxj,mini; prec maxv,
minv;
if(reMaxNecessary) {reMaxNecessary = FALSE; CortSetInitial();
```

```
ARRAYCOPY(bOrg[0],b[0],m);            ARRAYCOPY(cOrg[0],cR[m],n);}
CyclRepeatPrep();
while(TRUE) {maxv = - prec_MAX; maxj = -1; jLOOPL if(maxv <
cL[j])
{maxv = cL[j]; maxj = j;} jLOOPR if(maxv < cR[j]) {maxv = cR[j];
maxj = j;
}   if(maxv   <=   ZEROCUTCPV)   {FinMaxMF();   return   0;}
if(CyclRepeating(maxj))
{FinMaxMF(); return 1;} GenCol(maxj,m); mini = -1; minv =
prec_MAX;
for(i=0;i<m;i++) if(0 < ZeroPress(col[i])) if(b[i]/col[i] <
minv)
{minv = b[i]/col[i]; mini = i;} if(mini != -1) {if(b[mini]<0)
b[mini]=0;
GenRow(mini,jcol); rowR[jcol] = col[irow]; Pivot(TRUE);} else
{if(maxj<m)
{cL[maxj]    =    ZeroPress(cL[maxj]);}    else    {cR[maxj]    =
ZeroPress(cR[maxj]);}}}
} int CTMakeFeasible() {int i, mini; prec minv; int rtCond = 0;
CyclRepeatPrep(); while(TRUE) {mini = 0; minv = b[0];
for(i=1;i<m;i++)
if(b[i] < minv) {minv = b[i]; mini = i;} if(0<=ZeroPress(minv))
{FlushbOrgPrep();      for(i=0;i<m;i++)      FlushbOrgNote(i);
FlushbOrgFin();
FinMaxMF();    return    rtCond;}    if(CyclRepeating(u[mini]))
{CTReMax();
return   2;}   if((PivotRowOut(mini,   TRUE)))   {if(!rtCond)
{CTInvertMatrix();
rtCond = 1;} else {CTReMax(); return 2;}}}} void CTIncb(prec
facIn,
NEXTs& bSupl, prec& facOut, int& rtCond) {int i, ii, mini; prec
minv;
prec temp; NEXTs nextObOld; prec bInc[CORTMAX_M1]; facOut =
facIn;
rtCond = 0; iLOOPs(bSupl) if(ZeroPress(bSupl.val[i]) < 0) {temp
= -bOrg[i]
/bSupl.val[i]; if(temp < facOut) facOut = temp;} if(facOut <=
ZEROCUTCPV)
{facOut = 0; return;} CyclRepeatPrep(); ZEROOUT(bInc[0],m1);
for(ii=0;
ii<m;ii++) {iLOOPs(bSupl) bInc[ii] += B[ii][i] * bSupl.val[i];}
do {minv = 0; mini = -1; iLOOPs(nextOb) if(ZeroPress(bInc[i]) <
minv)
{minv   =   bInc[i];   mini   =   i;}   if(mini   !=   -1)
{if(!CyclRepeating(mini))
{if(PivotRowOut(mini, FALSE, bInc)) {facOut = 0; rtCond = 1;
return;}}
else {facOut = 0; rtCond = 2; return;}}} while(mini != -1);
for(i=0;i<m;
i++) if(ZeroPress(bInc[i]) < 0) {temp = -b[i]/bInc[i]; if(temp <
facOut)
{facOut = temp;}} if(!vMVClear) NextCopy(nextOb,nextObOld);
```

```
NextClear(next0b); for(i=0;i<m;i++) {b[i] += facOut * bInc[i];
if(b[i]
<= ZEROCUTSTO) b[i] = 0; if(b[i] <= ZEROCUTNEXT0b)
NextInsert(next0b,i);}
if(!vMVClear && !NextEqual(next0b,next0bOld)) ClearvMV();
FlushbOrgPrep();
iLOOPs(bSupl) {prec bOrgOld = bOrg[i]; bOrgBak[i] = bOrg[i];
bOrg[i]
+= facOut * bSupl.val[i]; if(bOrg[i] <= ZEROCUTSTO) bOrg[i] = 0;
if(bOrg[i] != bOrgOld) FlushbOrgNote(i);} FlushbOrgFin(); dgood
= FALSE;}
int CTReMax() {CortSetInitial(); ARRAYCOPY(bOrg[0],b[0],m);
ARRAYCOPY(cOrg[0],cR[m],n); return (CTMaximize());}
void Pivot(BOOL bUpdate, BOOL BrUpdate/*=FALSE*/, prec*
pbpbaseb/*=NULL*/,
prec* pbpcurb/*=NULL*/) {int i,j; prec factor[CORTMAX_M1];
NextjInsert(u[irow]); NextjDelete(jcol); u[irow] = jcol;
uref[jcol]
= irow; ClearvMV(); if(jcol<m) {for(i=0;i<m1;i++) B[i][jcol] =
0.0;
B[irow][jcol] = 1.0;} else cR[jcol] = 0.0; NextClear(nextTemp);
jLOOPL if(B[irow][j]) NextInsert(nextTemp,j); if(pbpbaseb)
pbpbaseb[m] =0;
if(pbpcurb) pbpcurb[m] =0; for(i=0;i<m1;i++) if(i != irow)
if(col[i])
{factor[i] = - col[i]/col[irow]; jLOOPs(nextTemp) B[i][j] +=
B[irow][j]
* factor[i]; if(bUpdate) b[i] += b[irow] * factor[i];
if(pbpbaseb)
pbpbaseb[i] += pbpbaseb[irow] * factor[i]; if(pbpcurb)
pbpcurb[i]
+= pbpcurb[irow] * factor[i];} jLOOPR cR[j] += rowR[j] * (-
col[m]
/col[irow]); jLOOPs(nextTemp) B[irow][j] /= col[irow];
if(bUpdate) b[irow]
/= col[irow]; if(BrUpdate) {if(m<=jcol) {iLOOPs(next0b)
B[i][jcol] = 0.0;
B[irow][jcol] = 1.0;} NextClear(nextTemp); jLOOPR if(B[irow][j])
NextInsert(nextTemp,j); iLOOPs(next0b) if(i != irow) if(col[i])
jLOOPs(nextTemp) B[i][j] += B[irow][j] * factor[i];
jLOOPs(nextTemp)
B[irow][j] /= col[irow];} if(pbpbaseb) pbpbaseb[irow] /=
col[irow];
if(pbpcurb) pbpcurb[irow] /= col[irow]; pivotCount++;} void
Pivot(int i,
int j, BOOL bUpdate) {GenCol(j,m); GenRow(i,mn);
Pivot(bUpdate);}
int PivotRowOut(int outRow, BOOL bUpdate, prec* pbpcurb
/*=NULL*/) {int j;
int minj = -1; prec minv=prec_MAX, temp; GenRow(outRow,mn);
jLOOPL if(ZeroPress(rowL[j]) < 0) {temp = cL[j]/rowL[j]; if(temp
< minv)
```

```
{minv = temp; minj = j;}} jLOOPR if(ZeroPress(rowR[j]) < 0)
{temp = cR[j]
/rowR[j]; if(temp < minv) {minv = temp; minj = j;}} if(minj == -
1)
return (1); GenCol(minj,irow); if(jcol<m) col[irow] =
rowL[jcol];
else col[irow] = rowR[jcol]; Pivot(bUpdate, FALSE, NULL,
pbpcurb);
return (0);} void FinMaxMF(BOOL checknegb /*=TRUE*/) {int i;
NEXTs next0bOld; if(checknegb) for(i=0;i<m;i++) if(b[i] <=
ZEROCUTSTO)
b[i] = 0; if(!vMVClear) NextCopy(next0b,next0bOld);
NextClear(next0b);
for(i=0;i<m;i++) if(b[i] <= ZEROCUTNEXT0b) NextInsert(next0b,
i);
if(!vMVClear && !NextEqual(next0b,next0bOld)) ClearvMV();}
void NextjInsert(int j) {int prevj = j - 1;
while(0<=prevj && !jnext[prevj]) prevj--; if(0<=prevj) {jnext[j]
= jnext[prevj]; jnext[prevj] = j;} else {jnext[j] = jstartL;
jstartL = j;}
if(j<jstartR && m <= j) jstartR = j; return;} void
NextjDelete(int j)
{int prevj = j - 1; while(0<=prevj && !jnext[prevj]) prevj--;
if(0<=prevj)
jnext[prevj] = jnext[j]; else jstartL = jnext[j]; if(j==jstartR)
jstartR = jnext[j]; jnext[j] =0;} void CyclRepeatPrep()
{cyclRepeatbm = prec_MAX; cyclRepeatIndex = -1;}
BOOL CyclRepeating(int index) {int j; if(cyclRepeatbm != b[m])
{cyclRepeatbm = b[m]; cyclRepeatIndex = -1; return FALSE;}
if(cyclRepeatIndex == -1) {cyclRepeatCount = mn + mn;
cyclRepeatIndex = index;
ARRAYCOPY(jnext[0],cyclRepeatjnext[0],mn);
return FALSE;} if(!cyclRepeatCount--) return TRUE;
if(cyclRepeatIndex != index) return FALSE; jLOOPL
if(!cyclRepeatjnext[j])
return FALSE; jLOOPR if(!cyclRepeatjnext[j]) return FALSE;
return TRUE;}
void CTbPlayBbyVec(prec* inb, prec* outb) {int i,ii;
ZEROOUT(outb[0],m);
for(ii=0;ii<m;ii++) for(i=0;i<m;i++) outb[ii] += B[ii][i] *
inb[i];}
void CTbPlayBbyVec(NEXTs& inbNext, prec* inb, prec* outb) {int
i,ii;
for(ii=0;ii<m;ii++) iLOOPs(inbNext) outb[ii] += B[ii][i] *
inb[i];}
void CTbPlayBbyVec(NEXTs& inbNext, prec* inb, int irow, prec&
outbele)
{int i; iLOOPs(inbNext) outbele += B[irow][i] * inb[i];}
int CTbPlayLoadbbOrg(prec* inb, prec* inbOrg, BOOL negb)
{ARRAYCOPY(inb [0], b [0], m); ARRAYCOPY(bOrg [0], bOrgBak[0],
m);
ARRAYCOPY(inbOrg[0], bOrg [0], m); dgood = FALSE; if(negb)
```

```
return CTMakeFeasible(); else {FinMaxMF(FALSE); return 0;}}
void  CTbPlayPrepMaxNegb()  {int  i,j;  iLOOPs(next0b)
{ZEROOUT(B[i][m],n);
j = u[i]; if(m<=j) B[i][j] = 1.0; jLOOPR B[i][j] =
GenCellR(i,j);}}
void  CTbPlayMaxNegb(prec*  pbpbaseb,  prec*  pbpcurb,  int&
bpcurmini,
prec  bpTipOver)  {BOOL  bpcolGen[CORTMAX_MN];  if(next0b.nele)
{BOOL cont;
do  {Pairint  bestCase(-1,-1,pbpcurb[bpcurmini]);  int  i,j;  int
i0row;
ZEROOUT(bpcolGen[m],n); cont = FALSE; LOOPs(next0b,i0row)
{prec  minv  =  prec_MAX;  int  minj  =  -1;  jLOOPL
if(ZeroPress(B[i0row][j])
< 0) if(cL[j]/B[i0row][j] < minv) {minv = cL[j]/B[i0row][j];
minj = j;}
jLOOPR  if(ZeroPress(B[i0row][j])  <  0)  if(cR[j]/B[i0row][j]  <
minv)
{minv = cR[j]/B[i0row][j]; minj = j;} if(minj != -1)
{prec  unitb  =  pbpcurb[i0row]/B[i0row][minj];  Pairint
curCase(i0row,minj,
unitb); if(m<=minj && !bpcolGen[minj]) {bpcolGen[minj] = TRUE;
for(i=0;i<m;
i++)  if(!next0b.use[i])  B[i][minj]  =  GenCellR(i,minj);
B[m][minj]
= cR[minj];} for(i=0;i<m;i++) if(i != i0row) {prec newb =
pbpcurb[i]
- B[i][minj] * unitb; if(newb < curCase.val) curCase.val =
newb;}
if(bestCase.val < curCase.val) {bestCase = curCase; cont =
TRUE;}}}
if(cont)  {bestCase.Get(irow,  jcol);
ARRAYCOPY(B[irow][m],rowR[m],n);
for(i=0;i<m1;i++)  col[i]  =  B[i][jcol];  Pivot(FALSE,  TRUE,
pbpbaseb,
pbpcurb); prec junk; ARRAYMIN(pbpcurb,bpcurmini,m,junk);}}
while(cont && pbpcurb[bpcurmini] < -bpTipOver);}}
BOOL CTShiftNeeded(int jShift) {return CTv2MV(jShift);}
void CTShiftBOrgIn(int iShift, prec facIn, prec& facOut) {facOut
= 1;
if(CTShiftNeeded(iShift)) {int i; int jShift = iShift;
prec  bOrgOrg  =  bOrg[iShift];  int  ct  =  0;
while(CTShiftNeeded(iShift)
&& ct != 25) {prec oldbOrg = bOrg[iShift]; ct++; if(ct==12)
{facIn = 0.98f; CTRoundAdjustment();} else if(ct==25) facIn =
0.0f;
facOut *= facIn; bOrgBak[iShift] = bOrg[iShift]; bOrg[iShift]
= facOut * bOrgOrg; prec bOrgnet = bOrg[iShift] - oldbOrg;
if(jnext[jShift])  {for(i=0;i<m;i++)  b[i]+=  B[i][jShift]  *
bOrgnet;}
else {i = uref[jShift]; b[i]+= bOrgnet;} CTMakeFeasible();}
dgood = FALSE;
```

```
ClearvMV();}} void CTClearGetvh() {ZEROOUT(cthQuant[0],n);
ZEROOUT(cthMC[0] ,n); ZEROOUT(ctvQuant[0],m); ZEROOUT(ctvMVs[0],
m);
ZEROOUT(ctvMVa[0],    m);    ClearvMV();}    prec    CTGetProfit()
{if(!dgood) Gend();
ctProfit   =   -b[m]/dFactorIn;   if(dolProfitAlso)   ctProfit   +=
DolGetProfit();
return  ctProfit;}  void  CTNoteDolProfitAlso()  {dolProfitAlso  =
TRUE;}
void  CTSetProfit(prec  profit)  {m  =  0;  dFactorIn  =  1;  dgood  =
TRUE; b[m]
= - profit;}
#define bOUT(i) ((prec)(b[i]*hScale[u[i]]/hScale[mn]))
#define jbOUT(j) ((prec)(b[uref[j]]*hScale[j] /hScale[mn]))
prec  CTGethQuant(int  jr)  {int  j  =  m  +  jr;  if(jnext[j])
cthQuant[jr] = 0;
else cthQuant[jr] = jbOUT(j); return cthQuant[jr];}
prec   CTGetRawProdQuant(int   jProd)   {int   j   =   m   +   jProd;
if(jnext[j])
return    0;    else    return    ZeroPress(b[uref[j]]);}    void
CTGethData(int jr,
int nOrder, int iStart/*=-1*/) {int j = m + jr; int i; switch
(nOrder)
{case 1: {cthMC[jr] = 0; iLOOPc(aCol[jr]) cthMC[jr] += ctvMVs[i]
* tableuOrg[i][jr]; break;} case 2: {if(jnext[j]) cthMC[jr]
= -(MincElement(j)/cFactorIn[j]) + CTGetcOrg(jr); else cthMC[jr]
=   CTGetcOrg(jr);   break;}   case   3:   {cthMC[jr]   =   0;
iLOOPc(aCol[jr])
if(iStart  <=  i)  {cthMC[jr]  +=  ctvMVs[i]  *  tableuOrg[i][jr];}
break;}
default:   {}}   CTGethQuant(jr);}   int   CTGetNumOb()   {return
nextOb.nele;}
int CTIsZerob(int i_brow) {return nextOb.use[i_brow];}
BOOL CTv2MV(int i_brow) {int i,j; j = i_brow; iLOOPs(nextOb)
if(ZeroPress(GenCellL(i,j)))          {int          j;          jLOOPL
if(ZeroPress(GenCellL(i,j)))
return TRUE; jLOOPR if(ZeroPress(GenCellR(i,j))) return TRUE;}
return FALSE;} void CTGetvQData(int i) {if(jnext[i]) ctvQuant[i]
= bOrg[i]
/bFactorIn[i];   else   ctvQuant[i]   =   bOrg[i]/bFactorIn[i]   -
jbOUT(i);}
void  CTGetvDataFin(int  i)  {if(ctvMVs[i]  <=  0)  {ctvMVs[i]  =  0;
ctvMVa[i]
=   0;}   else   {ctvMVa[i]   =   -(MaxcElement(i)/cFactorIn[i]);
if(ctvMVa[i] <= 0)
ctvMVa[i] = 0;} CTGetvQData(i); ctvMVcur[i] = TRUE;}
void  CTGetvData(int  i)  {vMVClear  =  TRUE;  if(ctmProd  <=  i)
{ctvMVs[i]
=   -(MincElement(i)/cFactorIn[i]);   CTGetvDataFin(i);}   else
{for(i=0;
i<ctmProd;i++)  if(ZeroPress(bOrg[i])  &&  !CTGetRawProdQuant(i))
{ctvMVs[i]
```

```
= 0;} else {ctvMVs[i] = -(MincElement(i)/cFactorIn[i]);}
for(i=0;
i<ctmProd;i++) CTGetvDataFin(i);} vMVClear = FALSE;} prec
CTGetb(int i)
{return bOUT(i);} prec CTGetbWT(int i) {return b[i];}
prec CTGetbOrgBak(int i) {return bOrgBak[i]/bFactorIn[i];}
prec CTGetbOrg(int i) {return bOrg[i]/bFactorIn[i];}
prec CTGetbOrgWT(int i) {return bOrg[i];} prec CTGetc(int j)
{if(j<m)
return cL[j]/cFactorIn[j]; else return cR[j]/cFactorIn[j];}
prec CTGetcOrg(int jr) {return cOrg[jr]/cFactorIn[m+jr];} void
ClearvMV()
{if(!vMVClear) {ZEROOUT(ctvMVcur[0],m); vMVClear = TRUE;}}
long double CTGetPivotCount() {return pivotCount;} prec
MincElement(int j)
{int i; BOOL cont = TRUE; prec cell; while(cont) {cont = FALSE;
iLOOPs(next0b) {if(u[i]==j) cell = 1; else cell =
ZeroPress(GenCell(i,j));
if(0<cell) if(!PivotRowOut(i,FALSE)) cont = TRUE;}} if(j<m)
return cL[j];
else return cR[j];} prec MaxcElement(int j) {int i; BOOL cont =
TRUE;
prec cell; if(!jnext[j]) return 0; while(cont) {cont = FALSE;
iLOOPs(next0b) {cell = ZeroPress(GenCell(i,j)); if(cell<0)
if(!PivotRowOut(i,FALSE)) cont = TRUE;}} if(j<m) return cL[j];
else return cR[j];} void GetCortDim(int& getm, int& getn, int&
getmProd)
{getm = m; getn = n; getmProd = ctmProd;} CTstor::CTstor()
{level = -1;}
#define storArray(anchor, elements) ARRAYCOPY(::anchor, anchor,
elements)
#define storArray2d(anchor, maxi, maxj) ARRAYCOPY2d(::anchor,
maxi, maxj,\
anchor)
#define storScalar(scalar)\
scalar = ::scalar
#define storStruct(structx) STRUCTCOPY(::structx,structx)
void CTstor::Out(int setlevel) {level = setlevel; if(0<=level)
{storArray(u[0], m); storArray(cR[m], n); storScalar(jstartL);
storScalar(jstartR);                   storArray(jnext[0],mn+1);
storArray(uref[0],mn);
storArray2d(B,m1,m);} if(1<=level) {storArray(bOrg[0], m);
storStruct(next0b); storArray(b[0], m1); storScalar(dgood);}
if(2<=level)
{storArray(cOrg[0], n);} if(3<=level) {storArray(vScale[0], m1);
storArray(hScale[0], mn+1); storArray(bFactorIn[0], m);
storArray(cFactorIn[0],        mn);        storScalar(dFactorIn);
storArray2d(tableuOrg,
m1,n1);}} void CTstor::OutvMV() {storArray(ctvMVs [0], m);
storArray(ctvMVa [0], m); storArray(ctvMVcur[0], m);}
#undef storArray
#undef storArray2d
```

```
#undef storScalar
#undef storStruct
#define storArray(anchor, elements) ARRAYCOPY(anchor, ::anchor,
elements)
#define  storArray2d(anchor,  maxi,  maxj)  ARRAYCOPY2d(anchor,
maxi, maxj,\
::anchor)
#define storScalar(scalar)\
::scalar = scalar
#define storStruct(structx) STRUCTCOPY(structx,::structx)
void    CTstor::In()    {if(0<=level)    {storArray(u[0],    m);
storArray(cR[m], n);
storScalar(jstartL);                         storScalar(jstartR);
storArray(jnext[0],mn+1);
storArray(uref[0],mn); storArray2d(B,m1,m);} if(1<=level)
{storArray(bOrg[0], m); ARRAYCOPY(bOrg[0], bOrgBak[0], m);
storStruct(next0b);  storArray(b[0],  m1);  storScalar(dgood);}
if(2<=level)
{storArray(cOrg[0], n);} if(3<=level) {storArray(vScale[0], m1);
storArray(hScale[0], mn+1); storArray(bFactorIn[0], m);
storArray(cFactorIn[0],      mn);      storScalar(dFactorIn);
storArray2d(tableuOrg,
m1,n1);} if(level == 3) CTMakeLoadCreatedSpace(); if(level)
{vMVClear = FALSE; ClearvMV();}} void CTstor::InvMV()
{storArray(ctvMVs   [0],   m);   storArray(ctvMVa   [0],   m);
storArray(ctvMVcur[0]
, m);}
#undef storArray
#undef storArray2d
#undef storScalar
#undef storStruct
IMPLEMENT_SERIAL(CTstor, CObject,1) CTstorp::CTstorp()
{pCTstor = new CTstor;} CTstorp::~CTstorp() {JDELETE(pCTstor);}
void CTstorp::Out(int setlevel) {pCTstor->Out(setlevel);}
void CTstorp::OutvMV() {pCTstor->OutvMV();} void CTstorp::In()
{pCTstor->In();} void CTstorp::InvMV() {pCTstor->InvMV();}
IMPLEMENT_SERIAL(CTstorp, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]dol.h[~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
#include "HCol.h"
void  DolClear();  void  DolLoadBody(NEXTs&  jrcNextDolSet,  int
dolRowSet,
int dolColSet); void DolLoadPrice(prec *priceSet);
void    DolIncAllotment(int    jrc,    prec    increment,    BOOL
updaterowFac);
void DolBalance(); void DolWalk(BOOL& profitable, long& ctDown,
HColIn* pIns =NULL, HColIn* pIna =NULL);
void DolGoGTransfer(BOOL callByRW=FALSE); void DolRWbpFill(BPds
&bbs,
prec qt); void DolRidgeWalk(BOOL& profitable); BOOL IsDolCol(int
jx);
prec DolGetProfit(); prec DolGetGroupPayOut();
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]dol.cpp[~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
HCol headHCw [RCDTMAX_NRC]; HCol headHCb [RCDTMAX_NRC]; NEXTs
jrcNextHCw;
NEXTs jrcNextHCb; NEXTs jrcNextDol;
#define dolAct jrcNextDol.nele
long dolRow; long dolCol; prec dolPrice[NEXTMAX];
#define dolQuant qOrg[dolCol]
HCol dolHCb; HCol dolHCw;
#define RCDs (*dolHCw.pRCs)
#define RCDa (*dolHCw.pRCa)
HCol dolHCdp [CORTMAX_M1]; HColIn dolRWPull[CORTMAX_M1];
HColIn dolRWDP [CORTMAX_M1]; HColIn dolRWPullRev[CORTMAX_M1];
HColIn dolRWDPRev [CORTMAX_M1]; HColIn dolRWPullSource;
HColIn dolRWDPSource; HColIn dolFix; HColIn dolFixRev; PairMan
dolpm;
prec dolBus; BOOL rwDolAct; void DolClear()
{NextPrep(jrcNextDol, nrc);
dolRow = dolCol = -1; ZEROOUT(dolPrice[0], nrc);}
void DolLoadBody(NEXTs& jrcNextDolSet, int dolRowSet, int
dolColSet)
{int jrc; NEXTs iNext; NextPrep1(iNext, dolRowSet, mrc);
NextInsert(jrcNextDolSet, dolColSet); LOOPs(jrcNextDolSet, jrc)
{LoadGroupDP(iNext, jrc); if(jrc != dolColSet)
NextDelete(horNextb[dolRowSet], jrc);} dolRow = dolRowSet;
dolCol = dolColSet; NextCopy(jrcNextDolSet, jrcNextDol);}
void DolLoadPrice(prec *priceSet) {ARRAYCOPY(priceSet[0],
dolPrice[0],
nrc);} void DolIncAllotment(int jrc, prec increment, BOOL
updaterowFac)
{int ig = dolRow; RCG.allotment += increment; if(jrc != dolCol)
updaterowFac = FALSE; GenGroupFactor(ig,jrc,updaterowFac);
if(jrc != dolCol) GenGroupMV(ig, jrc);} void DolBalance()
{if(dolAct)
{int jrc; BOOL cont; do {cont = FALSE; GenGroupMV(dolRow,
dolCol);
if(rc[dolRow][dolCol].gmvs < 1.0 - TOLERANCE)
{dolHCw.SetSub(dolRow,
dolCol); dolHCw.SetAdd(dolRow, jrcNextDol.lo); DolGoGTransfer();
cont = TRUE;} else if(rc[dolRow][dolCol].gmva > 1.0 - TOLERANCE)
{LOOPs(jrcNextDol, jrc) if(!rc[dolRow][jrc].onCorner && jrc !=
dolCol)
{dolHCw.SetSub(dolRow, jrc); dolHCw.SetAdd(dolRow, dolCol);
DolGoGTransfer(); cont = TRUE; break;}}} while(cont &&
ZeroPress(dolBus,
0.001));}} void DolWalk(BOOL& profitable, long& ctDown,
HColIn* pIns /*=NULL*/, HColIn* pIna /*=NULL*/)
{dolpm.WalkInit();
do {dolHCw.PairFind(TRUE, pIns, pIna); if(dolHCw.Goodsa())
{dolHCw.PairNote(); DolGoGTransfer();}} while(dolHCw.Goodsa()
&& 0<--ctDown); profitable = dolpm.WalkProfitable();
GenNextMV(jrcNext);}
```

```
void DolGoGTransfer(BOOL callByRW/*=FALSE*/) {int i; int ig, ie,
jrc;
int igs, jrcs; int iga, jrca; dolHCw.GetSub(igs, jrcs);
dolHCw.GetAdd(iga,
jrca); prec minFactor = 0; prec maxFactor = bigM; if(callByRW)
{minFactor = (RCDa.dirPut ? rwSliceMin * dolQuant : rwSliceMin /
RCDa.dedra); maxFactor = (RCDa.dirPut ? prec_MAX : rwSliceMax /
RCDa.dedra);}      if((!prSetInterNext[RCShc.rSet][RCAhc.rSet]))
{GtoGTransfer(igs, jrcs, iga, jrca, FALSE, callByRW, minFactor,
maxFactor);} else {if(GDecMax(igs,jrcs) < maxFactor) maxFactor =
GDecMax(igs,jrcs); if(GIncMax(iga,jrca) < maxFactor) maxFactor =
GIncMax(iga,jrca);          BOUNDP(minFactor,          maxFactor);
NextClear(bpiNext);
iLOOPs((*(prSetUnionNext[RCDs.rSet][RCDa.rSet])))
{NextInsert(bpiNext, i); ZEROOUT(bpFacVar[i][0],nrc);} jrc =
jrcs; ig = igs; if(ig != dolRow || jrc == dolCol) ieLOOP
bpFacVar[ie][jrc] = TRUE; jrc = jrca; ig = iga; if(ig != dolRow
|| jrc == dolCol) ieLOOP bpFacVar[ie][jrc] = TRUE;
BPreadybase(); BPlay(DolRWbpFill, minFactor, maxFactor);
IncAllotment(igs,jrcs,-dolBus, FALSE); IncAllotment(iga,jrca,
dolBus, FALSE); BPFin(); if(!callByRW && !ZeroPress(dolBus,
0.001)) dolHCw.PairBlock();}} void DolRWbpFill(BPds &bbs,prec
qt) {int i, ig, ie, jrc; int igs, jrcs; int iga, jrca;
dolHCw.GetSub(igs, jrcs); dolHCw.GetAdd(iga, jrca);
ARRAYCOPY(bpbase.bOrg[0],bbs.bOrg[0],mrc); jrc = jrcs; ig = igs;
if(ig != dolRow || jrc == dolCol) ieLOOP {bbs.bOrg[ie] *=
RCG.factor - RCG.dedrs * qt; if(bbs.bOrg[ie] < 0) {bbs.bOrg[ie]
= 0;}} jrc = jrca; ig = iga; if(ig != dolRow || jrc == dolCol)
ieLOOP {bbs.bOrg[ie] *= RCG.factor + RCG.dedra * qt;}
iLOOPs(bpiNext) bbs.bOrg[i] *= potentialCTwt[i]; dolBus = bbs.e
= qt;} void DolRidgeWalk(BOOL& profitable) {int ig, jrc; int
igSink =-1; int jrcSink=-1; profitable = rwProfitable = FALSE;
rwbestProfit = CTGetProfit(); rwbestImage.Out();
rwpm.WalkInit(); LOOPs(jrcNextDol, jrc) if(jrc != dolCol)
{igSink = dolRow; jrcSink = jrc; dolHCw.SetAdd(igSink, jrcSink);
break;} if(igSink == -1) {return;} ig = dolRow;
LOOPs(jrcNextDol, jrc) if(jrc != jrcSink && jrc != dolCol) {prec
qt = RCG.allotment; DolIncAllotment(jrc, -qt, FALSE);
DolIncAllotment(jrcSink, qt, FALSE);} GenRowFactor(dolRow);
rwsliceTrigger = 0; while(RCDa.allotment < dolQuant - TOLERANCE)
{if(dolHCw.SubFind(TRUE, igSink, jrcSink) == -1) break; prec
lastQt = RCDa.allotment; DolGoGTransfer(TRUE); if(rwsliceTrigger
< RCDa.allotment && !iLTimer.Elapse(iLrwTime)) {int rtCond2;
prec qt = RCDa.allotment; DolIncAllotment(jrcSink, -qt, FALSE);
rwsliceTrigger = RCDa.allotment + rwSliceFac * dolQuant;
roundAdjustmentOK[dolCol] = FALSE; GenNextMV(jrcNext);
ATLManager(rtCond2); dolHCw.SetAdd(igSink, jrcSink);
roundAdjustmentOK[dolCol] = TRUE; DolIncAllotment(jrcSink, qt,
FALSE); rwsliceTrigger = RCDa.allotment + rwSliceFac *
dolQuant;} if(rwbestProfit < CTGetProfit()) {rwProfitable =
TRUE; rwbestProfit = CTGetProfit(); rwbestImage.Out();}}
rwbestImage.In(); GenNextMV(jrcNext); if(rwProfitable) {BOOL
```

```
rtCond2;          ATLManager(rtCond2);          profitable          =
rwpm.WalkProfitable();}}    BOOL    IsDolCol(int    jx)    {return
jrcNextDol.use[jx];}   prec   DolGetProfit()   {prec   sum   =   0;
if(dolAct) {int ig = dolRow; int jrc; LOOPs(jrcNextDol, jrc) sum
+= RCG.allotment;} return sum;} prec DolGetGroupPayOut() {return
dolQuant - DolGetProfit();}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]ffp.h[~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
#ifndef FFP_h
class FFP : public CObject {DECLARE_SERIAL(FFP);
public:          prec*          startfp[RCDTMAX_NRC+1];          prec
factor[RCDTMAX_NRC+1];
prec potential; int nele; public: FFP(); ~FFP(); public: void
Init();
void     NoteRC(ResConduit&     rc,     BOOL     include);     prec
GetFactor_etal();
void Serialize(CArchive& ar);};
#define FFP_h
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]ffp.cpp[~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
#include "stdafx.h"
#include "jtools.h"
#include "cort.h"
#include "rcdt.h"
#include "FFP.h"
#define rfBasew factor[0]
FFP::FFP() {Init();} FFP::~FFP() {} void FFP::Init() {rfBasew =
nele = 1;}
void  FFP::NoteRC(ResConduit&  rc,  BOOL  include)  {if(include)
{startfp[nele]
=   &rc.dedrs;   factor[nele]   =   rc.factor;   rc.pFactorDep   =
&factor[nele];
nele++;}   else   {rfBasew   *=   rc.factor   ;   rc.pFactorDep   =
&rc.factor;}}
prec FFP::GetFactor_etal() {int j; prec fore[RCDTMAX_NRC+2];
prec back[RCDTMAX_NRC+2]; fore[0] = rfBasew; for(j=1;j<nele;j++)
fore[j]
= fore[j-1] * factor[j]; back[nele] = 1.0; for(j=nele-1;j;j--)
back[j]
= back[j+1] * factor[j]; for(j=1;j<nele;j++) {prec butFactor =
fore[j-1]
* back[j+1]; prec* pField = startfp[j]; prec dedrs = *pField++;
prec   dedra   =   *pField++;   *pField++   =   dedrs   *   butFactor   *
potential;
*pField = dedra * butFactor * potential;} return fore[nele-1];}
void FFP::Serialize(CArchive& ar) {CObject::Serialize(ar);}
IMPLEMENT_SERIAL(FFP, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]hcol.h[~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
#ifndef HCol_h
#define HCOLSIZE (CORTMAX_M1 * RCDTMAX_NRC)
```

```
class RCFilter; class HColIn : public CObject
{DECLARE_SERIAL(HColIn);
public: BOOL in[HCOLSIZE]; long inCt; BOOL nele; public:
HColIn();
~HColIn(); public: void Init(int nEleSet, BOOL allIn); void
In(int k);
void Ex(int k); void Reverse(); HColIn& operator+(HColIn& s);
HColIn& operator-(HColIn& s); HColIn& operator=(HColIn& s);
BOOL operator==(HColIn& r); BOOL operator!=(HColIn& r);
void Serialize(CArchive& ar);};
class HCol : public CObject {DECLARE_SERIAL(HCol);
public: ResConduit* pRC[HCOLSIZE]; long indexi[HCOLSIZE];
long indexj[HCOLSIZE]; long nele; long kBests; long kBesta;
ResConduit* pRCs; ResConduit* pRCa; BOOL notefinPend; BOOL
block[HCOLSIZE]
[HCOLSIZE]; long prematureBlockCT; short
pairCt[HCOLSIZE][HCOLSIZE];
public: HCol(); ~HCol(); void NoteInit(); void Note(ResConduit*
pRCNote,
int iNote, int jNote); void NoteDelete(ResConduit* pRCNote);
void NoteFin(); void Load(HCol& sour, HColIn* pIna=NULL);
void Load(HCol& sour, RCFilter& rcf); void BlockClear(); void
Clearsa();
void Clears(); void Cleara(); int Adjudges(); int Adjudgea();
BOOL Goodsa(); BOOL Goods(); BOOL Gooda(); int Findk(ResConduit*
pRCFind);
int Findk(int iFind, int jFind); BOOL Has(int ig, int jrc);
void GenColMV(); int GetFixCt(); prec GetSumAllot(RCFilter&
rcf);
prec GetSumAllot(); void ApportionDo(prec aquant, BOOL
genRowFactor);
void Apportion(prec aquant, BOOL genRowFactor);
void RoundAdjustment(prec aquant, BOOL genRowFactor);
void InInit(HColIn& hColIn, BOOL allIn); void Ex(HColIn& hColIn,
int iEx,
int jEx); void Ex(HColIn& hColIn, ResConduit* pRC);
void In(HColIn& hColIn, int iIn, int jIn); void In(HColIn&
hColIn,
ResConduit* pRC); void SetIn(HColIn& hColIn, RCFilter& rcf);
long PairFind(BOOL genMCMV, HColIn* pIns =NULL, HColIn* pIna
=NULL);
void PairNote(); void PairBlock(); void PairBlock(int igs, int
iga,
int jrcs, int jrca); int GetFUse(HColIn* pIn =NULL);
int SubFind(BOOL genMCMV, HColIn* pIns =NULL);
int SubFindtw(HColIn* pIns =NULL); int SubFind(BOOL genMCMV, int
igBut,
int jrcBut); void GetSub(int& igs, int& jrc); void SetSub(int
igs,
int jrc); BOOL IsSub(int igs, int jrc); void SubExclude(HColIn&
HColIn);
```

```
long AddFind(BOOL genMCMV, HColIn* pIna =NULL); void GetAdd(int&
igs,
int& jrc); void SetAdd(int igs, int jrc); BOOL IsAdd(int igs,
int jrc);
void AddExclude(HColIn& HColIn); void GetCord(int k, int& igs,
int& jrc);
void    SetsubBlk(BOOL    cond,    HColIn*    pIn    =NULL);    void
Serialize(CArchive& ar)
;};
#define prcLOOPhc(x) for(k=0, prc=x.pRC[k]; k < x.nele; prc =
x.pRC[++k])
#define   prcLOOPhcigjrc(x)   for(k=0,   prc=x.pRC[k],   ig   =
x.indexi[k],\
jrc  =  x.indexj[k];  k  <  x.nele;  prc=x.pRC[++k],  ig  =
x.indexi[k],\
jrc = x.indexj[k])
#define HCol_h
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]hcol.cpp[~~~~~~~~~~~~~~~~~~~~~~
~~~~~~
#include "stdafx.h"
#include "jtools.h"
#include "cort.h"
#include "rcdt.h"
#include "HCol.h"
#include "RCFilter.h"
HColIn::HColIn() {Init(0, TRUE);} HColIn::~HColIn() {}
void HColIn::Init(int nEleSet, BOOL allIn) {nele = nEleSet;
SPREAD(in,
nele, allIn); inCt = 0;} void HColIn::In(int k) {if(!in[k])
{in[k] = TRUE;
inCt++;}} void HColIn::Ex(int k) {if(in[k]) {in[k] = FALSE;
inCt--;}}
void HColIn::Reverse() {inCt = nele - inCt; for(int k=0;k<nele;
k++) in[k]
= !(in[k]);} HColIn& HColIn::operator+(HColIn& s) {for(int
k=0;k<nele;k++)
if(s.in[k]) In(k); return *this;} HColIn& HColIn::operator-
(HColIn& s)
{for(int k=0;k<nele;k++) if(s.in[k]) Ex(k); return *this;}
HColIn& HColIn::operator=(HColIn& s) {nele = s.nele; inCt =
s.inCt;
ARRAYCOPY(s.in[0], in[0], nele); return *this;}
BOOL HColIn::operator==(HColIn& r) {if(inCt == r.inCt) return
TRUE;
else return FALSE;} BOOL HColIn::operator!=(HColIn& r)
{return !(*this == r);} void HColIn::Serialize(CArchive& ar)
{CObject::Serialize(ar);} IMPLEMENT_SERIAL(HColIn, CObject,1)
#define okUses(x) (!pIns || pIns->in[x])
#define okUsea(x) (!pIna || pIna->in[x])
#define okUse(x) (!pIn || pIn ->in[x])
#define LOOPprc for(k=0, prc=pRC[k]; k < nele; prc = pRC[++k])
```

```
HCol::HCol()     {NoteInit();}     HCol::~HCol()     {}     void
HCol::NoteInit()
{nele = 0; Clearsa(); BlockClear(); notefinPend = FALSE;}
void  HCol::Note(ResConduit*  pRCNote,  int  iNote,  int  jNote)
{pRC[nele]
= pRCNote; indexi[nele] = iNote; indexj[nele] = jNote; nele++;
notefinPend = TRUE;} void HCol::NoteDelete(ResConduit* pRCNote)
{int k = Findk(pRCNote); for(int i=k+1; i<nele; i++) {pRC[i-1] =
pRC[i];
indexi[i-1] = indexi[i]; indexj[i-1] = indexj[i];} nele--;
notefinPend = TRUE;} void HCol::NoteFin() {BlockClear();
notefinPend  =  FALSE;}  void  HCol::Load(HCol&  sour,  HColIn*
pIna/*=NULL*/)
{NoteInit();    for(int    k=0;k<sour.nele;k++)    if(okUsea(k))
Note(sour.pRC[k],
sour.indexi[k],      sour.indexj[k]);      NoteFin();}      void
HCol::Load(HCol& sour,
RCFilter&   rcf)   {HColIn   hColIn;   sour.SetIn(hColIn,   rcf);
Load(sour,
&hColIn);}  void  HCol::BlockClear()  {ZEROOUT2D(block,  nele,
nele);
for(int  k=0;k<nele;k++)  block[k][k]  =  TRUE;  ZEROOUT2D(pairCt,
nele, nele);
prematureBlockCT   =   0;}   void   HCol::Clearsa()   {Clears();
Cleara();}
void  HCol::Clears()  {kBests  =  -1;  pRCs  =  NULL;}  void
HCol::Cleara()
{kBesta = -1; pRCa = NULL;} int HCol::Adjudges()
{if(0<=kBests && kBests <nele) {pRCs = pRC[kBests];
if((pRCs->ir != 0 || !pRCs->onCorner) && !pRCs->subBlk) return
kBests;}
Clears(); return kBests;} int HCol::Adjudgea()
{if(0<=kBesta && kBesta <nele) {pRCa = pRC[kBesta];
if(pRCa->ir  !=  pRCa->nir)  return  kBesta;}  Cleara();  return
kBesta;}
BOOL  HCol::Goodsa()  {return  (Goods()  &&  Gooda());}  BOOL
HCol::Goods()
{return  (BOOL)  (pRCs);}  BOOL  HCol::Gooda()  {return  (BOOL)
(pRCa);}
int  HCol::Findk(ResConduit*  pRCFind)  {for(int  k=0;k<nele;k++)
if(pRC[k]
== pRCFind) return k; return -1;} int HCol::Findk(int iFind, int
jFind)
{for(int  k=0;k<nele;k++)  if(indexi[k]  ==  iFind  &&  indexj[k]  ==
jFind)
return  k;  return  -1;}  BOOL  HCol::Has(int  ig,  int  jrc)  {return
(Findk(ig,
jrc) != -1);} void HCol::GenColMV() {for(int k=0;k<nele;k++)
GenGroupMV(indexi[k], indexj[k]);} int HCol::GetFixCt() {int k,
ct=0;
ResConduit* prc; LOOPprc if(prc->type == rctFix) ct++; return
ct;}
```

```
prec HCol::GetSumAllot(RCFilter& rcf) {int k; ResConduit* prc;
prec sum = 0; LOOPprc if(rcf.Pass(*prc)) sum += prc->allotment;
return sum;} prec HCol::GetSumAllot() {RCFilter rcf;
rcf.SetAll();
return GetSumAllot(rcf);} prec curAlloc[HCOLSIZE]={0};
void HCol::ApportionDo(prec aquant, BOOL genRowFactor) {int k;
ResConduit* prc; prec sumAlloc = 0; prec sumOver = 0; prec
sumUnder = 0;
prec under[HCOLSIZE]={0}; for(k=0;k<nele;k++) sumAlloc +=
curAlloc[k];
if(!ZeroPress(sumAlloc)) sumAlloc = 1; for(k=0;k<nele;k++)
curAlloc[k]
= aquant * (curAlloc[k]/sumAlloc);
LOOPprc {prec cap = prc->rstop[prc->nir]; if(cap <= curAlloc[k])
{sumOver += curAlloc[k] - cap; curAlloc[k] = cap;} else
{under[k]
= cap - curAlloc[k]; sumUnder += under[k];}} BOUNDP(sumOver,
sumUnder);
LOOPprc {if(under[k]) curAlloc[k] += sumOver *
(under[k]/sumUnder);
prc->allotment = curAlloc[k]; GenGroupFactor(indexi[k],
indexj[k],
genRowFactor);}} void HCol::Apportion(prec aquant, BOOL
genRowFactor)
{int k; ResConduit* prc; LOOPprc curAlloc[k] = __min(prc-
>rstop[prc->nir],
aquant); ApportionDo(aquant, genRowFactor);}
void HCol::RoundAdjustment(prec aquant, BOOL genRowFactor) {int
k;
ResConduit* prc; LOOPprc curAlloc[k] = prc->allotment;
ApportionDo(aquant,
genRowFactor);} void HCol::InInit(HColIn& hColIn, BOOL allIn)
{hColIn.Init(nele, allIn);} void HCol::Ex(HColIn& hColIn, int
iEx,
int jEx) {int k = Findk(iEx, jEx); hColIn.Ex(k);}
void HCol::Ex(HColIn& hColIn, ResConduit* pRC) {int k =
Findk(pRC);
hColIn.Ex(k);} void HCol::In(HColIn& hColIn, int iIn, int jIn)
{int k = Findk(iIn, jIn); hColIn.In(k);} void HCol::In(HColIn&
hColIn,
ResConduit* pRC) {int k = Findk(pRC); hColIn.In(k);}
void HCol::SetIn(HColIn& hColIn, RCFilter& rcf) {int k;
ResConduit* prc;
InInit(hColIn, FALSE); LOOPprc if(rcf.Pass(*prc)) In(hColIn,
indexi[k],
indexj[k]);} long HCol::PairFind(BOOL genMCMV, HColIn* pIns
/*=NULL*/,
HColIn* pIna /*=NULL*/) {int fUses = GetFUse(pIns);
int fUsea = GetFUse(pIna); int fUse = __min(fUses, fUsea);
if(fUse == -1)
{Clearsa(); return kBests;} kBests = fUses; kBesta = fUsea;
```

```
if((!pIns || !(pIns->inCt)) && (!pIna || !(pIna->inCt)))
{for(int k=fUse;
k<nele; k++) {if(genMCMV) GenGroupMV(indexi[k], indexj[k]);
if(pRC[k]
->gmvs < pRC[kBests]->gmvs) kBests = k; if(pRC[k]->gmva >
pRC[kBesta]
->gmva) kBesta = k;}} else {for(int k=fUse; k<nele; k++)
{if(genMCMV)
GenGroupMV(indexi[k], indexj[k]); if(pRC[k]->gmvs < pRC[kBests]
->gmvs && okUses(k)) kBests = k; if(pRC[k]->gmva > pRC[kBesta]
->gmva && okUsea(k)) kBesta = k;}} if(block[kBests][kBesta]
|| !okUses(kBests) || !okUsea(kBesta)) {prec mvs [HCOLSIZE];
prec mva [HCOLSIZE]; prec bestVal = 0; for(int k=fUse; k<nele;
k++)
{mvs[k] = okUses(k) ? pRC[k]->gmvs : prec_MAX/2; mva[k] =
okUsea(k)
? pRC[k]->gmva : -1;} for(int ks=fUse; ks<nele; ks++) for(int
ka=fUse;
ka<nele; ka++) if(bestVal < mva[ka] - mvs[ks] && !block[ks][ka])
{bestVal = mva[ka] - mvs[ks]; kBests = ks; kBesta = ka;}}
if(!block[kBests][kBesta] && okUses(kBests) && okUsea(kBesta))
{Adjudges()
; Adjudgea(); if(Goodsa()) {prec val = pRCa->gmva - pRCs->gmvs;
if(ZeroPress(val) <= 0) Clearsa();}} else {Clearsa();} return
kBests;}
void HCol::PairNote() {if(++pairCt[kBests][kBesta] == 80)
{PairBlock();
prematureBlockCT++;}} void HCol::PairBlock()
{block[kBests][kBesta]= TRUE;
} void HCol::PairBlock(int igs, int iga, int jrcs, int jrca)
{int is = Findk(igs, jrcs); int ia = Findk(iga, jrca);
block[is][ia]
= TRUE;} int HCol::GetFUse(HColIn* pIn /*=NULL*/) {for(int
k=0;k<nele;k++)
if(okUse(k)) return k; return -1;} int HCol::SubFind(BOOL
genMCMV,
HColIn* pIns /*=NULL*/) {int fUse = GetFUse(pIns); if(fUse == -
1)
{Clears(); return kBests;} kBests = fUse; for(int k=fUse;
k<nele; k++)
if(okUses(k)) {if(genMCMV) GenGroupMV(indexi[k], indexj[k]);
if(pRC[k]
->gmvs < pRC[kBests]->gmvs) kBests = k;} if(okUses(kBests))
{Adjudges();}
else {Clears();} return kBests;}
int HCol::SubFindtw(HColIn* pIns /*=NULL*/) {int fUse =
GetFUse(pIns);
if(fUse == -1) {Clears(); return kBests;} kBests = fUse; for(int
k=fUse;
k<nele; k++) if(okUses(k)) {if(pRC[k]->twgmvs < pRC[kBests]-
>twgmvs)
kBests = k;} if(okUses(kBests)) {Adjudges();} else {Clears();}
```

```
return kBests;} int HCol::SubFind(BOOL genMCMV, int igBut, int
jrcBut)
{if(!nele) {Clears(); return kBests;} HColIn includeList;
InInit(includeList, TRUE); Ex(includeList, igBut, jrcBut);
int rt = SubFind(genMCMV, &includeList); return rt;}
void HCol::GetSub(int& igs, int& jrc) {igs = indexi[kBests];
jrc = indexj[kBests];} void HCol::SetSub(int igs, int jrc)
{kBests = Findk(igs, jrc); Adjudges();} BOOL HCol::IsSub(int
igs, int jrc)
{int k = Findk(igs, jrc); return (kBests == k);}
void HCol::SubExclude(HColIn& HColIn) {HColIn.Ex(kBests);}
long HCol::AddFind(BOOL genMCMV, HColIn* pIna /*=NULL*/)
{int fUse = GetFUse(pIna); if(fUse == -1) {Cleara(); return
kBesta;}
kBesta = fUse; for(int k=fUse; k<nele; k++) if(okUsea(k))
{if(genMCMV)
GenGroupMV(indexi[k], indexj[k]); if(pRC[k]->gmva > pRC[kBesta]-
>gmva)
kBesta = k;} if(okUsea(kBesta)) {Adjudgea();} else {Cleara();}
return kBesta;} void HCol::GetAdd(int& igs, int& jrc)
{igs = indexi[kBesta]; jrc = indexj[kBesta];} void
HCol::SetAdd(int igs,
int jrc) {kBesta = Findk(igs, jrc); Adjudgea();} BOOL
HCol::IsAdd(int igs,
int jrc) {int k = Findk(igs, jrc); return (kBesta == k);}
void HCol::AddExclude(HColIn& HColIn) {HColIn.Ex(kBesta);}
void HCol::GetCord(int k, int& igs, int& jrc) {igs = indexi[k];
jrc = indexj[k];} void HCol::SetsubBlk(BOOL cond, HColIn* pIn
/*=NULL*/)
{for(int k=0;k<nele;k++) if(okUse(k)) ::SetsubBlk(indexi[k],
indexj[k],
cond);} void HCol::Serialize(CArchive& ar)
{CObject::Serialize(ar);}
IMPLEMENT_SERIAL(HCol, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]jtools.h[~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~
#ifndef JTOOLS_H
#include <memory.h>
#include <limits.h>
#include <math.h>
#include <float.h>
#define TOLERANCE_DBL ((0.001f)/(999.999f))
#define TOLERANCE_LDB ((long double)(0.001f)/((long
double)(999.999f)))
#define bigM 1.e50
#define TOLERANCE TOLERANCE_DBL
#define prec double
#define precabs fabs
#define precsqrt sqrt
#define xprec_MAX ((double) DBL_MAX/2.0f)
#define xprec_SORTMAX ((double) DBL_MAX)
#define DOUBLEPREC
```

```
#define CORTMAX_M1 32
#define CORTMAX_N 16
#define nrSetMAX 64
#define RCDTMAX_NRC 8
#define RCDTMAX_PT 9
#define CORTMAX_MN (CORTMAX_M1+CORTMAX_N)
#define NEXTMAX __max(CORTMAX_MN, nrSetMAX)
#define DIM1(anchor) (sizeof(anchor) /sizeof(anchor[0]))
#define NEW(ptr,spec) {ptr = new spec;}
#define JDELETE(ptr) {if(ptr != NULL) {delete ptr; ptr = NULL;}}
#define JDELETEA(ptr) {if(ptr != NULL) {delete [] ptr; ptr =
NULL;}}
#define PT2TOCRECT(p1,p2) CRect(p1.x, p1.y, p2.x, p2.y)
#define NUMTEST(number) {long double z_xtemp = number;}
#define ZEROOUT(element, number) {memset((void*) &(element), 0,
(number)\
*sizeof(element)); NUMTEST(element);}
#define ZEROOUTSTRUCT(strct) {memset((void*) &(strct), 0,
(sizeof(strct))\
;}
#define ZEROOUT2D(anchor, maxi, maxj) {NUMTEST(anchor[0][0]);
for(int i=0;\
i<maxi;i++) {ZEROOUT(anchor[i][0],maxj);}}
#define ARRAYCOPY(sour,dest,n) {memcpy((void*) (&dest), (void*)
&(sour),\
(n)* sizeof(sour)); NUMTEST(sour); NUMTEST(dest);}
#define ARRAYCOPY2d(anchor, maxi, maxj, dest) {for(int
i=0;i<maxi;i++)\
ARRAYCOPY(anchor[i][0], dest[i][0], maxj);}
#define STRUCTCOPY(sour,dest) {memcpy((void*) (&dest), (void*)
&(sour),\
sizeof(sour));}
#define SPURTP(variable, limit) if(variable < limit) variable =
limit;
#define BOUNDP(variable, limit) if(variable > limit) variable =
limit;
#define pCAST(name, p) ((name) (p))
#define JIsKindOf(name, p) (p && p-
>IsKindOf(RUNTIME_CLASS(name)))
#define JIsA(name, p) (p && p->GetRuntimeClass() ==
RUNTIME_CLASS(name))
#define MAPi for(i=0; i<pMap->mRow2-1; i++)
#define MAPj for(j=0; pMap->GetpEle(i+1,j,pCd); j++)
#define pMAPj0 (pMap->GetpEle(i+1,0))
#define pMAPj1 (pMap->GetpEle(i+1,1))
#define pMAPj2 (pMap->GetpEle(i+1,2))
#define MAPgv pMap->GetpEle(i+1,j)->VFetch()
#define pMAPt0 (pMap->GetpEle(0,0))
#define pMAPt1 (pMap->GetpEle(0,1))
#define pMAPt2 (pMap->GetpEle(0,2))
#include "NEXTs.h"
```

```
struct NEXTc {int use[NEXTMAX+1]; prec val[NEXTMAX]; int cur,
mele;};
#define iLOOPs(ns) for(i=ns.lo;i<ns.mele;i=ns.use[i])
#define jLOOPs(ns) for(j=ns.lo;j<ns.mele;j=ns.use[j])
#define LOOPs(ns,k) for(k=ns.lo;k<ns.mele;k=ns.use[k])
#define LOOPs_del(ns,k) for(k=NextFollow(ns,-1);k<ns.mele;\
k=NextFollow(ns,k))
#define LOOPs_limitIter(ns,k,maxIterate) for(k=ns.lo,\
ns.nstemp = maxIterate; k<ns.mele && 0<ns.nstemp; k=ns.use[k],\
ns.nstemp--)
#define RESUMELOOPs(ns, k) for(k=NextFollow(ns,k-1);k<ns.mele;\
k=NextFollow(ns,k))
#define iLOOPc(nc) for(i=nc.use[nc.cur=0];nc.cur<nc.mele;\
i=nc.use[++nc.cur])
#define jLOOPc(nc) for(j=nc.use[nc.cur=0];nc.cur<nc.mele;\
j=nc.use[++nc.cur])
#define LOOPc(nc,k) for(k=nc.use[nc.cur=0];nc.cur<nc.mele;\
k=nc.use[++nc.cur])
#define SQrt2xPI 2.501342788
class Min2; class Max2; class Meaner; class JTimer; extern prec
prec_MAX;
extern prec prec_SORTMAX; void RndSeedScat(long& rndSeed);
void Rnd(long& rndSeed); double ZeroPress(double value,
double tolerance = TOLERANCE_DBL);
long double ZeroPress(long double value,
long double tolerance = TOLERANCE_LDB); BOOL IsEqual(double q1,
double q2,
double tolerance = TOLERANCE_DBL); BOOL IsEqual(long double q1,
long double q2, long double tolerance = TOLERANCE_LDB);
BOOL IsEqualComP(double q1, double q2, double tolerance =
TOLERANCE_DBL);
BOOL IsEqualComP(long double q1, long double q2,
long double tolerance = TOLERANCE_LDB); prec Interpolate(prec
x0, prec y0,
prec x1, prec y1, prec xhat); prec Interpolate(prec x0, prec y0,
prec x1,
prec y1, prec x2, prec y2, prec xhat); BOOL Divisible(double
numer,
double demon); BOOL Divisible(long double numer, long double
demon);
void NextPrep(NEXTs& ns, int nelement); void NextPrep1(NEXTs&
ns, int k ,
int nelement =-1); void NextPrep(NEXTc& nc, int nelement, BOOL);
void NextClear(NEXTc& nc); void NextClear(NEXTs& ns);
void NextInsert(NEXTs& ns, int k); void NextInsert(NEXTs& ns,
int k,
prec elementValue); void NextDelete(NEXTs& ns, int k);
void NextCompress(NEXTs& ns, BOOL transVal, NEXTc& nc);
void NextFill(NEXTs& ns, int nelement); void NextReverse(NEXTs&
sour,
NEXTs& dest); int NextSortComp(const void* ele_a, const void*
ele_b);
```

```
void NextSort(NEXTs& ns, int multi =1);
int NextSortCompIV(const void* ele_a, const void* ele_b);
void NextSortIV(NEXTs& ns, BOOL ascSort=TRUE); void
NextCopy(NEXTs& sour,
NEXTs& dest, BOOL transVal =FALSE); BOOL NextEqual(NEXTs& nsa,
NEXTs& nsb,
BOOL testValalso =FALSE); void NextAdd(NEXTs& result, NEXTs&
nsr);
void NextSub(NEXTs& result, NEXTs& nsr); BOOL NextOverlap(NEXTs&
nsl,
NEXTs& nsr); void NextIntersect(NEXTs& result, NEXTs& nsl,
NEXTs& nsr);
void NextUnion(NEXTs& result, NEXTs& nsl, NEXTs& nsr);
void NextAround(NEXTs& ns, int& k); int NextFollow(NEXTs& ns,
int k);
void Abort(CString& message); void Abort(char* message);
#define SPREAD(anchor,max,value) {int zi; for(zi=0;zi<max;zi++)
anchor[zi]\
= value; NUMTEST(anchor[0]);}
#define SPREAD2d(anchor,maxi, maxj ,value) {int zi, zj;
for(zi=0;zi<maxi;\
zi++) for(zj=0;zj<maxj;zj++) anchor[zi][zj] = value;
NUMTEST(anchor[0][0])\
;}
#define ARRAYMIN(anchor,index,maxindex,value) {index = 0;\
value = anchor[0]; for(int izt=1;izt<maxindex;izt++)
if(anchor[izt]\
< value) {value = anchor[izt]; index = izt;}}
class Pairint : public CObject {DECLARE_SERIAL(Pairint); public:
int i;
int j; prec val; BOOL bol; public: Pairint(); Pairint(int seti,
int setj);
Pairint(int seti, int setj, prec setv); void Load (int seti, int
setj);
void Load (int seti, int setj, prec setv); void Loadi(int seti);
void Loadj(int setj); void Get (int& geti, int& getj, prec&
getv);
void Get (int& geti, int& getj); void Geti (int& geti);
void Getj (int& getj); Pairint& operator=(const Pairint& s);
BOOL operator==(const Pairint& r); BOOL operator!=(const
Pairint& r);};
class Pairprec : public CObject {DECLARE_SERIAL(Pairprec);
public: prec x;
prec y; public: Pairprec(); Pairprec(prec setx, prec sety);
Pairprec& operator=(const Pairprec& s);};
class IndexPrec : public CObject {DECLARE_SERIAL(IndexPrec);
public: int ind; prec val; public: IndexPrec(); IndexPrec(int
setindex,
prec setvalue); void Load(int setindex, prec setvalue); int
GetIndex();
prec GetValue(); IndexPrec& operator=(const IndexPrec& s);
```

```
BOOL operator==(const IndexPrec& r); BOOL operator!=(const
IndexPrec& r);}
; class Min2 : public CObject {DECLARE_SERIAL(Min2); public:
IndexPrec e1;
IndexPrec e2; public: Min2(); void Init();
void Init(prec maxPossibleValue); BOOL Note(int index, prec&
value);
BOOL Note1st(int index, prec& value); BOOL Note2nd(int index,
prec& value)
; BOOL NoteImprovement(int index, prec& value); int GetBest();
int GetBestBut(int indexBut);};
class Max2 : public CObject {DECLARE_SERIAL(Max2); public:
IndexPrec e1;
IndexPrec e2; public: Max2(); void Init(); BOOL Note(int index,
prec& value); BOOL Note1st(int index, prec& value);
BOOL Note2nd(int index, prec& value); BOOL NoteImprovement(int
index,
prec& value); int GetBest(); int GetBestBut(int indexBut);};
#define JTOOLS_H
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]jtools.cpp[~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~
#include "stdafx.h"
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "jtools.h"
struct NextRow {int i; prec val;}; NextRow nextTable[NEXTMAX+2];
BOOL nextascSort; prec prec_MAX = xprec_MAX;
prec prec_SORTMAX = xprec_SORTMAX; void RndSeedScat(long&
rndSeed)
{rndSeed *= SQrt2xPI; if(!rndSeed) rndSeed = 15347991; for(int
j=0;j<25;
j++) {long m = 2147483647; long bb = rndSeed % 100;
rndSeed += bb * 1000000 + bb * 10000 + bb * 100 + bb;
if(rndSeed<0)
rndSeed += m; if(!rndSeed) rndSeed = bb + 123; Rnd(rndSeed);}}
void Rnd(long& rndSeed) {long a = 16807; long m = 2147423777;
long q = 124237; long r = 2836; long lo; long hi; long test;
hi = rndSeed/q; lo = rndSeed % q; test = a * lo - r * hi; if(0 <
test)
rndSeed = test; else rndSeed = test + m;} double
ZeroPress(double value,
double tolerance /*= TOLERANCE_DBL*/) {if(fabs(value) <
tolerance)
value = 0.0; return value;} long double ZeroPress(long double
value,
long double tolerance /*= TOLERANCE_LDB*/) {if(fabsl(value) <
tolerance)
value = 0.0; return value;} BOOL IsEqual(double q1, double q2,
double tolerance /*= TOLERANCE_DBL*/) {if(fabs(q1-q2) <
tolerance)
```

```
return TRUE; else return FALSE;} BOOL IsEqual(long double q1,
long double q2, long double tolerance /*= TOLERANCE_LDB*/)
{if(fabsl(q1-q2) < tolerance) return TRUE; else return FALSE;}
BOOL IsEqualComP(double q1, double q2,
double tolerance /*= TOLERANCE_DBL*/) {if(fabs(q1-q2) <
tolerance)
return TRUE; else {q1 = fabs(q1); q2 = fabs(q2); if(q2<q1)
Swap(q1,q2);
double div = 1.0f - (q1/q2); if(div < tolerance) return TRUE;
else return FALSE;}} BOOL IsEqualComP(long double q1, long
double q2,
long double tolerance /*= TOLERANCE_LDB*/) {if(fabsl(q1-q2) <
tolerance)
return TRUE; else {q1 = fabsl(q1); q2 = fabsl(q2); if(q2<q1)
Swap(q1,q2);
long double div = 1.0f - (q1/q2); if(div < tolerance) return
TRUE;
else return FALSE;}} prec Interpolate(prec x0, prec y0, prec x1,
prec y1,
prec xhat) {prec s, yhat; if(x0 == x1) return(y0+y1)/2; s = (y1-
y0)
/(x1-x0); yhat = y0 + s * (xhat-x0); return yhat;}
prec Interpolate(prec x0, prec y0, prec x1, prec y1, prec x2,
prec y2,
prec xhat) {int i, ii, j; prec yhat; long double w[3][4];
w[0][1] = x0;
w[1][1] = x1; w[2][1] = x2; for(i=0;i<3;i++) w[i][2] = w[i][1] *
w[i][1];
w[0][3] = y0; w[1][3] = y1; w[2][3] = y2; for(i=1;i<3;i++)
for(j=1;j<4;
j++) w[i][j] -= w[0][j]; if(fabsl(w[1][1]) < fabsl(w[2][1]))
for(j=1;j<4;
j++) Swap(w[1][j],w[2][j]); for(ii=1;ii<3;ii++)
{for(j=ii+1;j<4;j++)
if(Divisible(w[ii][j],w[ii][ii])) w[ii][j] /= w[ii][ii];
else return prec_MAX; for(i=ii+1;i<3;i++) for(j=ii+1;j<4;j++)
w[i][j]
-= w[i][ii] * w[ii][j];} for(ii=2;0<ii;ii--) for(i=0;i<ii;i++)
w[i][3]
-=w[i][ii] * w[ii][3]; yhat = (prec) (w[0][3] + w[1][3] * xhat +
w[2][3]
* xhat * xhat); return yhat;} BOOL Divisible(double numer,
double demon)
{int ne, de; if(!(demon)) return FALSE; frexp(numer,&ne);
frexp(demon,&de)
; if((ne-de) < DBL_MAX_EXP-3) return TRUE; else return FALSE;}
BOOL Divisible(long double numer, long double demon) {int ne,
de;
if(!(demon)) return FALSE; frexpl(numer,&ne); frexpl(demon,&de);
if((ne-de) < LDBL_MAX_EXP-3) return TRUE; else return FALSE;}
void NextPrep(NEXTs& ns, int nelement) {ns.mele = nelement;
```

```
ZEROOUT(ns.use[0],ns.mele); ns.use[ns.mele] = ns.lo = ns.hi =
ns.mele;
ns.nele = 0;} void NextPrep1(NEXTs& ns, int k , int nelement
/*=-1*/)
{if(nelement == -1) nelement = k+1; NextPrep(ns, nelement);
NextInsert(ns,
k);} void NextPrep(NEXTc& nc, int nelement, BOOL) {nc.mele =
nelement;
if(nelement == 0) nelement = 1;} void NextClear(NEXTc& nc)
{nc.mele = 0;}
void NextClear(NEXTs& ns) {int i = ns.lo; while(i < ns.mele)
{int temp = i; i = ns.use[i]; ns.use[temp] = 0;} ns.use[ns.mele]
= ns.lo = ns.hi = ns.mele; ns.nele = 0;} void NextInsert(NEXTs&
ns, int k)
{int i; if(ns.hi < k) {ns.use[ns.hi] = k; ns.use[k] = ns.mele;
ns.hi = k;
ns.nele++;} else if(k < ns.lo) {ns.use[k] = ns.lo; ns.lo = k;
if(!ns.nele++) ns.hi = k;} else iLOOPs(ns) if(i < k && k <
ns.use[i])
{ns.use[k] = ns.use[i]; ns.use[i] = k; ns.nele++; break;}}
void NextInsert(NEXTs& ns, int k, prec elementValue)
{NextInsert(ns,k);
ns.val[k] = elementValue;} void NextDelete(NEXTs& ns, int k)
{int i;
if(ns.lo == k) {ns.lo = ns.use[k]; ns.use[k] = 0; ns.nele--;
if(!ns.nele)
ns.hi = ns.mele;} else iLOOPs(ns) if(ns.use[i] == k) {ns.use[i]
= ns.use[k]; ns.use[k] = 0; if(ns.hi == k) ns.hi = i; ns.nele--;
break;}}
void NextCompress(NEXTs& ns, BOOL transVal, NEXTc& nc) {int i,
n=0;
NextPrep(nc,ns.nele,transVal); iLOOPs(ns) {nc.use[n] = i;
if(transVal)
nc.val[n] = ns.val[i]; n++;}} void NextFill(NEXTs& ns, int
nelement)
{int i; NextPrep(ns,nelement); for(i=0;i<nelement;i++)
NextInsert(ns,i);}
void NextReverse(NEXTs& sour, NEXTs& dest) {int i;
NextPrep(dest,
sour.mele); for(i=0;i<sour.mele;i++) if(!sour.use[i])
NextInsert(dest,i);}
int NextSortComp(const void* ele_a, const void* ele_b)
{if(*((prec *)
ele_a) < *((prec *) ele_b)) return -1; if(*((prec *) ele_a) ==
*((prec *)
ele_b)) return 0; else return 1;} void NextSort(NEXTs& ns,
int multi /*=1*/) {int i, n; prec sortarray[NEXTMAX]; if(ns.nele
<= 1)
return; n = 0; iLOOPs(ns) sortarray[n++] = (multi == 1) ?
ns.val[i]
: - ns.val[i]; qsort(sortarray,n,sizeof(prec),NextSortComp); n =
0;
```

```
iLOOPs(ns) ns.val[i] = (multi == 1) ? sortarray[n++] : -
sortarray[n++];}
int NextSortCompIV(const void* ele_a, const void* ele_b)
{if(((NextRow *)
ele_a)->val < ((NextRow *) ele_b)->val) if(nextascSort) return -
1;
else return 1; if(((NextRow *) ele_a)->val == ((NextRow *)
ele_b)->val)
return 0; else if(nextascSort) return 1; else return -1;}
void NextSortIV(NEXTs& ns, BOOL ascSort/*=TRUE*/) {int i, p, n =
0;
if(ns.nele <= 1) return; iLOOPs(ns) {nextTable[n].i = i;
nextTable[n]
.val = ns.val[i]; n++;} nextTable[n].i = ns.mele; nextascSort =
ascSort;
qsort(nextTable,n,sizeof(nextTable)/(NEXTMAX+2),NextSortCompIV);
p = ns.lo = nextTable[0].i; for(i=0;i<n;i++) {ns.use[p] =
nextTable[i+1]
.i; p = nextTable[i+1].i;}} void NextCopy(NEXTs& sour, NEXTs&
dest,
BOOL transVal /*=FALSE*/) {int i; dest.lo = sour.lo; dest.hi =
sour.hi;
dest.mele = sour.mele; dest.nele = sour.nele;
ARRAYCOPY(sour.use[0],
dest.use[0],sour.mele+1); if(transVal) iLOOPs(sour) dest.val[i]
= sour.val[i];} BOOL NextEqual(NEXTs& nsa, NEXTs& nsb,
BOOL testValalso /*=FALSE*/) {int i;
if(nsa.lo != nsb.lo || nsa.hi != nsb.hi || nsa.mele != nsb.mele
|| nsa.nele != nsb.nele) return FALSE; iLOOPs(nsa) if(nsa.use[i]
!= nsb.use[i]) return FALSE; if(testValalso) iLOOPs(nsa)
if(nsa.val[i] != nsb.val[i]) return FALSE; return TRUE;} void
NextAdd(NEXTs& result, NEXTs& nsr) {int i; iLOOPs(nsr)
NextInsert(result,i);} void NextSub(NEXTs& result, NEXTs& nsr)
{int i; iLOOPs(nsr) NextDelete(result,i);} BOOL
NextOverlap(NEXTs& nsl, NEXTs& nsr) {int i; iLOOPs(nsr)
if(nsl.use[i]) return TRUE; return FALSE;} void
NextIntersect(NEXTs& result, NEXTs& nsl, NEXTs& nsr) {int i;
NextPrep(result, nsl.mele); iLOOPs(nsl) if(nsr.use[i])
NextInsert(result,i);} void NextUnion(NEXTs& result, NEXTs& nsl,
NEXTs& nsr) {NextCopy(nsl, result); NextAdd(result, nsr);} void
NextAround(NEXTs& ns, int& k) {if(ns.use[k]) k = ns.use[k]; else
k = NextFollow(ns,k); if(k==ns.mele) k = ns.lo;} int
NextFollow(NEXTs& ns, int k) {if(k<0) return ns.lo; k++;
if(ns.mele <= k) return ns.mele; while(!ns.use[k]) k++; return
k;} void Abort(CString& message) {exit(1);} void Abort(char*
message) {CString ss = message; Abort(ss);} Pairint::Pairint()
{} Pairint::Pairint(int seti, int setj) {Load(seti,setj);}
Pairint::Pairint(int seti, int setj, prec setv)
{Load(seti,setj,setv);} void Pairint::Load (int seti, int setj)
{i = seti; j = setj;} void Pairint::Load (int seti, int setj,
prec setv) {i = seti; j = setj; val = setv;} void
Pairint::Loadi(int seti) {i = seti;} void Pairint::Loadj(int
```

```
setj) {j = setj;} void Pairint::Get (int& geti, int& getj, prec&
getv) {geti = i; getj = j; getv = val;} void Pairint::Get (int&
geti, int& getj) {geti = i; getj = j;} void Pairint::Geti (int&
geti) {geti = i;} void Pairint::Getj (int& getj) {getj = j;}
Pairint& Pairint::operator=(const Pairint& s) {if(&s != this) {i
= s.i; j = s.j; val = s.val; bol = s.bol;} return *this;} BOOL
Pairint::operator==(const Pairint& r) {if(i == r.i && j == r.j)
return TRUE; else return FALSE;} BOOL Pairint::operator!=(const
Pairint& r) {if(*this == r) return FALSE; else return TRUE;}
IMPLEMENT_SERIAL(Pairint, CObject,1) Pairprec::Pairprec() {}
Pairprec::Pairprec(prec setx, prec sety) {x = setx; y = sety;}
Pairprec& Pairprec::operator=(const Pairprec& s) {if(&s != this)
{x = s.x; y = s.y;} return *this;} IMPLEMENT_SERIAL(Pairprec,
CObject,1) IndexPrec::IndexPrec() {} IndexPrec::IndexPrec(int
setindex, prec setvalue) {Load(setindex,setvalue);} void
IndexPrec::Load(int setindex, prec setvalue) {ind = setindex;
val = setvalue;} int IndexPrec::GetIndex() {return ind;} prec
IndexPrec::GetValue() {return val;} IndexPrec&
IndexPrec::operator=(const IndexPrec& s) {if(&s != this) {ind =
s.ind; val = s.val;} return *this;} BOOL
IndexPrec::operator==(const IndexPrec& r) {if(ind== r.ind)
return TRUE; else return FALSE;} BOOL
IndexPrec::operator!=(const IndexPrec& r) {if(*this == r) return
FALSE; else return TRUE;} IMPLEMENT_SERIAL(IndexPrec, CObject,1)
Min2::Min2() {Init();} void Min2::Init() {e1.Load(-1,prec_MAX);
e2.Load(-1,prec_MAX);} void Min2::Init(prec maxPossibleValue)
{e1.Load(-1,maxPossibleValue); e2.Load(-1,maxPossibleValue);}
BOOL Min2::Note(int index, prec& value) {IndexPrec
w(index,value); if(w.val < e1.val) {e2 = e1; e1 = w; return
TRUE;} else if(w.val < e2.val) {e2 = w; return TRUE;} else
return FALSE;} BOOL Min2::Note1st(int index, prec& value)
{IndexPrec w(index,value); if(w.val < e1.val) {e1 = w; return
TRUE;} else return FALSE;} BOOL Min2::Note2nd(int index, prec&
value) {IndexPrec w(index,value); if(w.val < e2.val && w != e1)
{e2 = w; return TRUE;} else return FALSE;} BOOL
Min2::NoteImprovement(int index, prec& value) {IndexPrec
w(index,value); if(w == e1) {e1 = w; return TRUE;} else if(w ==
e2) {e2 = w; if(e2.val < e1.val) SWAP(e1,e2,IndexPrec); return
TRUE;} else return Note(index, value);} int Min2::GetBest()
{return e1.GetIndex();} int Min2::GetBestBut(int indexBut)
{if(e1.ind != indexBut) return e1.GetIndex(); else return
e2.GetIndex();} IMPLEMENT_SERIAL(Min2, CObject,1) Max2::Max2()
{Init();} void Max2::Init() {e1.Load(-1,-prec_MAX); e2.Load(-1,-
prec_MAX);} BOOL Max2::Note(int index, prec& value) {IndexPrec
w(index,value); if(w.val > e1.val) {e2 = e1; e1 = w; return
TRUE;} else if(w.val > e2.val) {e2 = w; return TRUE;} else
return FALSE;} BOOL Max2::Note1st(int index, prec& value)
{IndexPrec w(index,value); if(w.val > e1.val) {e1 = w; return
TRUE;} else return FALSE;} BOOL Max2::Note2nd(int index, prec&
value) {IndexPrec w(index,value); if(w.val > e2.val && w != e1)
{e2 = w; return TRUE;} else return FALSE;} BOOL
Max2::NoteImprovement(int index, prec& value) {IndexPrec
```

```
w(index,value); if(w == e1) {e1 = w; return TRUE;} else if(w ==
e2) {e2 = w; if(e2.val > e1.val) SWAP(e1,e2,IndexPrec); return
TRUE;} else return Note(index, value);} int Max2::GetBest()
{return e1.GetIndex();} int Max2::GetBestBut(int indexBut)
{if(e1.ind != indexBut) return e1.GetIndex(); else return
e2.GetIndex();} IMPLEMENT_SERIAL(Max2, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]locator.h[~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
#ifndef Locator_h
class JCellD; class JCellM; class JCellA;
class Locator : public CObject {DECLARE_SERIAL(Locator);
public: JCellA* pwA; WTYPE wType; int ir0; int ir1; int
dgetjcolSet;
public: Locator(); Locator(Locator& ll); Locator(JCellA*
pwASet);
void Init(Locator& ll); void Init(JCellA* pwASet); void
NoteIr(int ir);
void Inc_ir0(); void Init_ir1(); void Inc_ir1(); int
GetRow0Count();
int GetRow1Count(); int GetGrandTotalRow1Count(); JCellD* p0(int
jCol);
JCellD* p1(int jCol); JCellM* p0Map(int jCol); JCellM* p1Map(int
jCol);
int GetRowID(int irowType, int jCol, CString& search);
void FindNote(int irowType, int jCol, CString& search);
JCellD* DGet(int irow, int jcol); void DGetSetCol(int jcol);
JCellD* DGet(int irow);};
#define Locator_h
#define LOCLOOP0(lc) for(lc.ir0=1 ; lc.ir0<lc.pwA->mSide;
lc.Inc_ir0())
#define LOCLOOP1(lc) for(lc.Init_ir1(); lc.ir1<lc.pwA->mSide;
lc.Inc_ir1()\
)
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]locator.cpp[~~~~~~~~~~~~~~~~~~~
~~~~~~~~~
#include "stdafx.h"
#include "jtools.h"
#include "Locator.h"
Locator::Locator() {} Locator::Locator(Locator& ll)
{Init(ll.pwA);}
Locator::Locator(JCellA* pwASet) {Init(pwASet);}
void Locator::Init(Locator& ll) {Init(ll.pwA);}
void Locator::Init(JCellA* pwASet) {ir0 = ir1 = -1; pwA =
pwASet;
if(pwASet) wType = pwA->wType; dgetjcolSet = -1;}
void Locator::NoteIr(int ir) {if(pwA->pSide[ir]->irowType == 0)
{ir0 = ir;
ir1 = -1;} else {ir1 = ir; do ir--; while(pwA->pSide[ir]-
>irowType == 1);
ir0 = ir;}} void Locator::Inc_ir0() {do ir0++;
while(ir0<pwA->mSide && pwA->pSide[ir0]->irowType); ir1 = -1;}
```

```
void Locator::Init_irl() {irl = ir0; Inc_irl();} void
Locator::Inc_irl()
{irl++; if(pwA->mSide <= irl || pwA->pSide[irl]->irowType==0)
irl = pwA->mSide;} int Locator::GetRow0Count() {int ct = 0;
for(int i=1;
i<pwA->mSide;i++) if(pwA->pSide[i]->irowType == 0) ct++; return
ct;}
int Locator::GetRow1Count() {int ct = 0; for(int i=ir0+1;i<pwA-
>mSide;i++)
if(pwA->pSide[i]->irowType == 1) ct++; else break; return ct;}
int Locator::GetGrandTotalRow1Count() {int ct = 0; for(int i=1;
i<pwA->mSide;i++) if(pwA->pSide[i]->irowType == 1) ct++; return
ct;}
JCellD* Locator::p0(int jCol) {return (JCellD*) pwA->pSide[ir0]
->pBody[jCol];} JCellD* Locator::p1(int jCol) {return (JCellD*)
pwA->pSide[irl]->pBody[jCol];} JCellM* Locator::p0Map(int jCol)
{return (JCellM*) pwA->pSide[ir0]->pBody[jCol];}
JCellM* Locator::p1Map(int jCol) {return (JCellM*) pwA-
>pSide[irl]
->pBody[jCol];} int Locator::GetRowID(int irowType, int jCol,
CString& search) {int i; for(i=1;i<pwA->mSide;i++) if(pwA-
>pSide[i]
->irowType == irowType) if(pCAST(JCellD*, pwA->pSide[i]-
>pBody[jCol])
->rdString == search) return i; return -1;}
void Locator::FindNote(int irowType, int jCol, CString& search)
{int i = GetRowID(irowType, jCol, search); NoteIr(i);}
JCellD* Locator::DGet(int irow, int jcol) {return (JCellD*)
pwA->pSide[irow]->pBody[jcol];} void Locator::DGetSetCol(int
jcol)
{dgetjcolSet = jcol;} JCellD* Locator::DGet(int irow) {return
DGet(irow,
dgetjcolSet);} IMPLEMENT_SERIAL(Locator, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~]nexts.h[~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
#ifndef NEXTs_h
class NEXTs : public CObject {DECLARE_SERIAL(NEXTs);
public: int use[NEXTMAX+1]; prec val[NEXTMAX]; int lo, hi, nele,
mele;
int nstemp; public: public: NEXTs(); void Resetmele(int mNew);
void Serialize(CArchive& ar);};
#define NEXTs_h
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~]nexts.cpp[~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
#include "stdafx.h"
#include "jtools.h"
#include "NEXTs.h"
NEXTs::NEXTs() {} void NEXTs::Resetmele(int mNew) {for(int i=0;
i<mele;i++)
if(use[i] == mele) use[i] = mNew; mele = mNew;}
void NEXTs::Serialize(CArchive& ar) {CObject::Serialize(ar);}
```

```
IMPLEMENT_SERIAL(NEXTs, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]rcdt.h[~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
#ifndef RCDT_h
#include "Jtools.h"
class RCFilter; class HCol; class HColIn;
#define gmv_MAX bigM
#include "ResConduit.h"
struct resConduitRCstor {prec allotment;};
struct BPds {prec bOrg[CORTMAX_M1]; prec b [CORTMAX_M1]; prec e;
prec minv; BOOL negb; int adjust; prec bpestinc; prec netMV;};
enum twCOND {normalTWCond, abortChoke, abortPair};
enum twCASE {simplelink, splitnc, splitcyVer, splitcyHor,
gtogNullLink};
#define rcarcsMAX (RCDTMAX_NRC+3)
extern prec rcMVs [RCDTMAX_NRC]; extern prec rcMVa
[RCDTMAX_NRC];
extern prec rctwMVs [RCDTMAX_NRC]; extern prec rctwMVa
[RCDTMAX_NRC];
extern prec rcQuant [RCDTMAX_NRC]; extern prec
rcQuantFix[RCDTMAX_NRC];
extern prec rcPotMVs [CORTMAX_M1]; extern prec rcPotMVa
[CORTMAX_M1];
HCol* GetpDolHCw(); void RCDTPrep(int setmrc, int setnrc, int
setmProd);
void RCSetTolerance(); int GenrSetId(NEXTs& ieList); int
GetrSetId();
void LoadGroup(NEXTs& ieList, int jrc, ResConduit& grc);
void LoadGroupDP(NEXTs& ieList, int jrc, prec slope =1.0);
void LoadqOrg(prec* pData); void LoadPotential(prec* pData);
void SetrcTypeAsFix(int ig, int jrc, prec allotment);
BOOL IsGoodrcTypeAsFix(); rcType GetrcType(int ig, int jrc);
void SetrwMax(long iter, prec timeSet); void SetRWSlice(prec
sliceMin,
prec sliceMax, prec sliceFac); void GetRWSlice(prec& sliceMin,
prec& sliceMax, prec& sliceFac); prec GetqOrgwtIn(int jrc);
void IncAllotmentTest(int ig, int jrc, prec increment);
prec GetColSumAllotqOrgwtOt(int jrc); int GetColType(int jrc);
void RFiNextEqualPurge(NEXTs& ns); void SetRCType();
void SetMarginCtlW(BOOL seekVar, BOOL seeknonFix); void
SetMarginCtlB();
void SetMarginCtlFinNew(RCFilter& rcf); void
SetMarginCtlFinOld();
void FFPBuild(); void OrientPotentialxx(); void
GenGroupFactor(int ig,
int jrc, BOOL updaterowFactor); void GenRowFactor(int i);
void GenRowFactor(); void GenRowFactor(NEXTs& rfNext);
prec GetRowFactor(int i); void GenGroupMV(int ig, int jrc);
void GenNextMV(NEXTs& ns); void GenColMV(int jrc); void
GenPairMV(int ig1,
int ig2, int jrc); void PermBlockPair(int igs, int iga, int
jrcs,
```

```
int jrca); BOOL RCGetPermBlockPair(int igs, int iga, int jrc);
void GtoGForceThru(int igs, int iga, int jrc, prec quant);
void GtoGForceThru(int igs, int iga, int jrcs, int jrca, prec
quant);
void GtoGTransfer(int igs, int iga, int jrc, BOOL
callByLW=FALSE,
BOOL callByRW=FALSE, prec minFactor =0.0f, prec maxFactor
=bigM);
void GtoGTransfer(int igs, int jrcs, int iga, int jrca, BOOL,
BOOL callByRW=FALSE, prec minFactor =0.0f, prec maxFactor
=bigM);
int GetRowMinS(int jrc); int GetRowMinS(int igbut, int jrc);
prec GIncMax(int ig, int jrc); prec GDecMax(int ig, int jrc);
void IncAllotment(int ig, int jrc, prec increment, BOOL
updaterowFac);
void XctMVPre(); void XctMVGet(int i); void XctMVPost(); void
LoadCTbOrg()
; void RCRoundAdjustment(BOOL doWalk=TRUE); void
RCRoundAdjustmentwAW();
void MakeCTFeasible(); void IncCTb(prec facIn, NEXTs& bSupl,
prec& facOut)
; void LoadCTb_bOrg(prec* inb, prec* inbOrg, BOOL negb); void
BPPrep();
void BPreadybase(); int BPGenadjust(int i, prec bele);
int BPMinb(BPds& bp); void BP_BbyVec(BPds& bp); void
BPprepSmartBbybpcur()
; void BPSmartBbybpcur(BOOL reset, BOOL acceptPosAdjust);
void BPlay(void (genb)(BPds &bp, prec e), prec mine, prec maxe);
void BPFin(BOOL genRowFactor=TRUE); void AWPrep();
void AxisWalk(int& rtCond); void ExplodeWalk(int& rtCond); void
TWPrep();
void TWPrepWalkSession(); void TWSetMargins(); BOOL
TWisTopped(int iRow);
void TWGenitopNext(int lwControl); prec
TWGenNetFactor(ResConduit& rc,
prec& qbase); void TWNotePossiblyBetterO(int ig, int jrc);
void TWGetu(Pairint& v, int& uig, int& ujrc);
void TWIncPairintv(Pairint& v); prec TWufv(ResConduit& urc, prec
uqbase,
ResConduit& vrc, prec vqbase); prec TWufv(ResConduit& urc, prec
uqbase,
ResConduit& vrc, prec vqbase, prec vquant); prec
TWvfu(ResConduit& urc,
prec uqbase, ResConduit& vrc, prec vqbase); prec
TWvfu(ResConduit& urc,
prec uqbase, ResConduit& vrc, prec vqbase, prec uquant); void
TWGenMVs();
void TWGenMVa(); void TWPushThru(prec snqt); void
TWGenFactor(BPds& bbs,
BOOL noteanchorfactorBut =FALSE); int TWGentwMQuant();
void TWevalMV(BPds& bbs); void TWbpFill(BPds& bbs, prec snqt);
```

```
void TWSwapQin(); BOOL TWCrossHat(Pairint& p); void
TWLoadrcarcs(int iga,
int igs, int jrc); void TWCheckCleanCycle(); void
TWQuantSufficient();
void TWLoadbpiNext(); void TWLoadBPlayEquateMvMc(); void
TWDoBlockage();
void TWTryIteration(); void TWdoFundamentalTW(BOOL& get2ndpair,
BOOL& raCurrent); void TWdoCleanCycleTW(BOOL& get2ndpair);
void TWMaxsplitcyYield(void (FnEval)(Pairprec& coordinate));
void TWsplitcyVerYield(Pairprec& coordinate);
void TWsplitcyHorYield(Pairprec& coordinate); void
TWGensnqtMax();
void TopWalkExit(BOOL& profitable); void TopWalk(int& rtCond,
BOOL& profitable, int lwControl); BOOL TW0SplitGroup(); void
RWPrep();
void RWPrepWalkSession(); void RWbpFill(BPds &bbs,prec f);
void SetsubBlk(int ie, int jrc, BOOL cond, BOOL allElements
=TRUE);
void SwapsubBlk(HCol& hcol); void SetRWiHATBlocking(BOOL cond);
void RWDPDrag(int ipull, BOOL firstPass=FALSE);
void RidgeWalk(int& rtCond, BOOL& profitable, int ipull); void
LWPrep();
void LWSetFactor(prec setlwFactor); prec LWGetfacIn(); void
LWInitLat();
void LWRestoreLatCtMar(BOOL ctImageAlso=TRUE); void
LWRestoreOrgCtMar();
void LateralWalk(BOOL& profitable); void ATLManager(int&
rtCond);
void WalkReadyInteration(); void Walk(); void LoadHC(NEXTs&
jNext,
HCol& hcol); void MaxFreshPrep(BOOL weight);
void MaxFresh(BOOL weight=FALSE); void MaxPotential(prec
*potentialNew,
BOOL doWalk =TRUE); void WeightqOrg(); void MaxqOrg(prec
*qOrgNew);
void MaxPotentialqOrg(prec *potentialNew, prec *qOrgNew);
BOOL GetPermanentBlockPair(); void ConvDolToReal(int jrc, prec&
gmvs,
prec& gmva, prec& qtAllot); ResConduit* GetpGroup(int ig, int
jrc);
void GetGroupMV(int ig, int jrc, prec& gmvs, prec& gmva, prec&
qtAllot);
void GetGrouptwMV(int ig, int jrc, prec& twgmvs, prec& twgmva,
prec& qtAllot); void RCGetRowMC(int ie, BOOL infMC, int& rtCond,
prec& mc)
; BOOL RCGetResultVec(); class CKer : public CObject
{DECLARE_SERIAL(CKer);
public: int index[CORTMAX_M1 + 1]; prec val [CORTMAX_M1]; int
nele;
int mink; prec minv; int maxk; prec maxv; int klooper; BOOL
sorted;
public: CKer(); ~CKer(); void Init(); void Load(NEXTs& ns);
```

```
void Load(CKer& cks); int GetK(int indexSearch); void GenMin();
void GenMax(); void SortAscend(); void SortDescend();
void DeleteValso(int k);};
#define ckLOOPk(cker, k, iloop) for(k=0,iloop=cker.index[k];
k<cker.nele;\
iloop=cker.index[++k])
#define ckLOOP(cker, iloop) ckLOOPk(cker, cker.klooper, iloop)
class PairMan : public CObject {DECLARE_SERIAL(PairMan);
private: NEXTs jrcBestNext; BOOL bestEvalPend[RCDTMAX_NRC];
int bestigs [RCDTMAX_NRC]; int bestiga [RCDTMAX_NRC];
static CKer ckref [RCDTMAX_NRC]; CKer ckmin [RCDTMAX_NRC];
CKer ckmax [RCDTMAX_NRC]; int itjrc; int itks; int itka; int
igsTemp;
int igaTemp; int jrcTemp; static int pairMinCt; static prec
pairMinInc;
static int pairMaxCt; static prec blockMinInc; static int
blockMaxCt;
static prec walkMinInc; prec walkProfitBaseNote; prec
pairProfitBaseNote;
BOOL blockDone; prec blockDoneProfit; int blockDoneCt;
BOOL permBlock[CORTMAX_M1][CORTMAX_M1][RCDTMAX_NRC]; BOOL
permBlockClear;
struct {int count; BOOL blocked;} pm3d[CORTMAX_M1][CORTMAX_M1]
[RCDTMAX_NRC]; static int assigtok[CORTMAX_M1][RCDTMAX_NRC];
private: public: private: int GetK(CKer& ck, int ig, int jrc);
BOOL IsProfitable(prec oprofit, prec& minInc); void BlockFin();
public: static void LoadigPrep(); static void Loadig(int jrc,
NEXTs& igNext); static void SetTolerance(int prminCt, prec
prminInc,
int prmaxCt, prec bkminInc, int bkmaxCt, prec wkminInc);
static void GetTolerance(int& prminCt, prec& prminInc, int&
prmaxCt,
prec& bkminInc, int& bkmaxCt, prec& wkminInc); void
PermBlockInit();
PairMan(); void WalkInit(); void Clear(NEXTs&
jrcentertainSetNext);
void ItPrep(); void ItPrepCol(int jrc); void ItNoteSub(prec
value);
void ItNoteAdd(prec value); void ItDelSub(int ig, int jrc);
void ItDelAdd(int ig, int jrc); void ItUpdateValueSub(int ig,
int jrc,
prec newValue); void ItUpdateValueAdd(int ig, int jrc, prec
newValue);
void GetBestPair(int& getigs, int& getiga, int& getjrc, prec
&getvalue);
void GetBestPair(int& getigs, int& getiga, int& getjrc); void
PrePair();
int IncPairCount(int igs, int iga, int jrc); int
GetPairCount(int igs,
int iga, int jrc); void PostPair(int igs, int iga, int jrc);
void BlockPair(int igs, int iga, int jrc); void BlockPair(int
igs,
```

```
int iga, int jrc, prec curProfit); void Blockigs(int iga, int
jrc);
void Blockiga(int igs, int jrc); void PermBlockPair(int igs, int
iga,
int jrc); BOOL GetPermBlockPair(int igs, int iga, int jrc);
BOOL ProfitableSinceBlocked(); BOOL WalkProfitable();
void SetTemp(int setigs, int setiga, int setjrc);
void GetTemp(int& getigs, int& getiga, int& getjrc);};
class RCstor : public CObject {DECLARE_SERIAL(RCstor); private:
int level;
CTstor ctImage; resConduitRCstor rc[CORTMAX_M1][RCDTMAX_NRC];
prec potential [CORTMAX_M1]; prec qOrg [RCDTMAX_NRC]; prec
profit;
public: RCstor(); void Out(int setlevel=0); void OutProfit();
void In();
prec GetProfit();};
class RCstorp : public CObject {DECLARE_SERIAL(RCstorp);
private: RCstor* pRCstor; public: RCstorp(); ~RCstorp();
void Out(int setlevel=0); void OutProfit(); void In(); prec
GetProfit();};
#define RCDT_h
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~]rcdt.cpp[~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~
#include "stdafx.h"
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "cort.h"
#include "rcdt.h"
#include "dol.h"
#include "FFP.h"
#include "RCFilter.h"
prec rcMVs [RCDTMAX_NRC]; prec rcMVa [RCDTMAX_NRC];
prec rctwMVs [RCDTMAX_NRC]; prec rctwMVa [RCDTMAX_NRC];
prec rcQuant [RCDTMAX_NRC]; prec rcQuantFix[RCDTMAX_NRC];
prec rcPotMVs [CORTMAX_M1]; prec rcPotMVa [CORTMAX_M1];
NEXTs rSetieNext [nrSetMAX]; int nrSet; NEXTs* prSetUnionNext
[nrSetMAX]
[nrSetMAX] = {NULL}; NEXTs* prSetInterNext [nrSetMAX][nrSetMAX]
= {NULL};
NEXTs* prSetSubNext [nrSetMAX][nrSetMAX] = {NULL}; prec qOrg
[RCDTMAX_NRC]
; prec qOrgFix [RCDTMAX_NRC]; prec qOrgwtIn [RCDTMAX_NRC];
prec qOrgwtOt [RCDTMAX_NRC]; NEXTs jrcNext; NEXTs headNextw
[RCDTMAX_NRC];
NEXTs headNextb [RCDTMAX_NRC]; prec rowFactor[CORTMAX_M1];
FFP ffp [CORTMAX_M1]; prec rfBasew [CORTMAX_M1];
prec maxRowFac[CORTMAX_M1]; NEXTs irfNextb; NEXTs irfNextw;
NEXTs ipullNext; NEXTs ilwNext; NEXTs horNextw [CORTMAX_M1];
NEXTs horNextb [CORTMAX_M1]; Min2 minig[RCDTMAX_NRC];
```

```
Max2 maxig[RCDTMAX_NRC]; ResConduit rc[CORTMAX_M1][RCDTMAX_NRC];
int nrc,
mrc; int nid; long iLaw; long iLtw; long iLawCt; long iLtwCt;
long iLlwCt;
JTimer iLTimer; long rcmProd; prec iLggFactorTolerance;
prec iLggColumnTolerance[RCDTMAX_NRC]; long iLggCount; PairMan
iLpm;
NEXTs dpNext; prec potential [CORTMAX_M1]; prec
potentialCTwt[CORTMAX_M1];
BOOL permanentBlockPair; long rcRoundAdjustmentCt = 0;
BOOL roundAdjustmentOK[RCDTMAX_NRC];
#define jrcLOOP LOOPs(jrcNext,jrc)
#define igwLOOP LOOPs(headNextw[jrc],ig)
#define igbLOOP LOOPs(headNextb[jrc],ig)
#define RCG rc[ig][jrc]
#define RCE rc[ie][jrc]
#define ieLOOP for(ie=ig; ie<mrc;ie=rc[ie][jrc].ieDown)
#define RCS rc[igs][jrc]
#define RCA rc[iga][jrc]
#define RCShc rc[igs][jrcs]
#define RCAhc rc[iga][jrca]
#define gMAXABSORB (RCG.rstop[RCG.nir])
#define TOLERANCEQ TOLERANCE
#define TOLERANCEV TOLERANCE
#define ZEROCUTV ((prec) 0)
#define IsEqualQ(q1,q2) IsEqual(q1,q2,TOLERANCEQ)
#define IsEqualV(v1,v2) IsEqual(v1,v2,TOLERANCE)
#define ZeroPressQ(q) ZeroPress(q, TOLERANCEQ)
#define ZeroPressV(v) ZeroPress(v, TOLERANCE)
PairMan wkpm; BOOL initialLoad = FALSE; prec rowFactorTolerance
= 0.001f;
long ffpCt=0; long permBlockPairCt;
BOOL gtogTransferOKSwapDirection = TRUE; const int bpParaMax =
5;
const prec bpTipOver = TOLERANCE/10000.0; BOOL
bpFacVar[CORTMAX_M1]
[RCDTMAX_NRC]; BPds bpbase, bplo, bpcur, bphi; NEXTs bpiNext;
NEXTs bpirNext; NEXTs bpiTouchNext; PairMan awpm; NEXTs
bIncNext;
NEXTs jrcawNext; NEXTs itopNext; NEXTs jrctwNext;
NEXTs igtwNext[RCDTMAX_NRC]; Min2 rSetminj[nrSetMAX];
NEXTs twhorNextv [CORTMAX_M1]; NEXTs twrSetNext;
NEXTs rowrSetNext[CORTMAX_M1]; NEXTs pendrSetNext[nrSetMAX];
BOOL pendrSet[CORTMAX_M1][nrSetMAX]; NEXTs
gOverLapRowNext[CORTMAX_M1];
BOOL multinonLk; int nonLkct[CORTMAX_M1]; Pairint
rca[rcarcsMAX];
Pairint rcs[rcarcsMAX]; int iSplitVer; int iSplitHor; int
twnLink;
Pairint twChoke; PairMan twpm; prec snqtMax; long twFootPrt;
BOOL cleanCycle; Pairint strayG[2 * rcarcsMAX];
Pairint strayE[CORTMAX_M1 * RCDTMAX_NRC]; int nstrayG, nstrayE;
```

```
int nstrayEanchorStart; prec twToleranceClamp = TOLERANCE*0.45f;
twCASE twcase; twCOND twcond; prec anchorfactorBut[CORTMAX_M1];
NEXTs twGenFactorNext;
#define jrctwLOOP LOOPs(jrctwNext,jrc)
#define igtwLOOP LOOPs(igtwNext[jrc],ig)
#define RCU rc[uig][ujrc]
#define RCUe rc[ie][ujrc]
#define RCV rc[vig][vjrc]
#define RCVe rc[ie][vjrc]
#define RCof(pair) rc[pair.i][pair.j]
#define RCuDown rc[RCU.twuDownig][ujrc]
#define vEnd RCof(rcs[twnLink-1])
#define RCsph RCof(rca[iSplitHor])
#define RC2s RCof(rcs[iSplitHor-1])
int rwiHAT; NEXTs jrcrwNextBase; NEXTs jrcbusNext;
#define jrcbusLOOP LOOPs(jrcbusNext,jrc)
int sbig[RCDTMAX_NRC]; NEXTs sbiNext; int adig[RCDTMAX_NRC];
NEXTs adiNext;
#define RCs rc[sbig[jrc]][jrc]
#define RCa rc[adig[jrc]][jrc]
#define IGs sbig[jrc]
#define IGa adig[jrc]
#define ieLOOPs for(ie=sbig[jrc];ie<mrc;ie=rc[ie][jrc].ieDown)
#define ieLOOPa for(ie=adig[jrc];ie<mrc;ie=rc[ie][jrc].ieDown)
prec oldQuant[RCDTMAX_NRC]; prec oldMC [RCDTMAX_NRC];
prec bus [RCDTMAX_NRC]; prec rwSliceMin = 0.001f;
prec rwSliceMax = 0.250f; prec rwSliceFac = 0.001f; RCstor
rwbestImage;
BOOL rwProfitable; prec rwbestProfit; prec rwsliceTrigger;
int dpTieb [CORTMAX_M1][RCDTMAX_NRC]; NEXTs dpTieNext
[CORTMAX_M1];
NEXTs dpTieNextDol [CORTMAX_M1]; PairMan rwpm; long iLrw; prec
iLrwTime;
PairMan& lwpm = twpm; NEXTs jrclwNext; prec lwFactor = 0.999;
CTstor lwOrgCTimage; prec lwOrgProfit; prec lwOrgpotential
[CORTMAX_M1];
prec lwOrgpotentialCTwt[CORTMAX_M1]; CTstor lwLatCTimage;
prec lwLatpotential [CORTMAX_M1]; prec lwLatpotentialCTwt
[CORTMAX_M1];
NEXTs irowExplode;
#include "dol.cpp"
HCol* GetpDolHCw() {return &dolHCw;} void RCDTPrep(int setmrc,
int setnrc,
int setmProd) {int i, j; mrc = setmrc; nrc = setnrc; rcmProd =
setmProd;
for(i=0;i<mrc;i++) for(j=0;j<nrc;j++) ZEROOUTSTRUCT(rc[i][j]);
for(i=0;
i<mrc;i++) {NextPrep(horNextw[i],nrc);
NextPrep(horNextb[i],nrc);} for(j=0;
j<nrc;j++) {NextPrep(headNextw[j],mrc);
NextPrep(headNextb[j],mrc);}
```

```
NextPrep(irfNextb,      mrc);      NextPrep(irfNextw,       mrc);
NextPrep(irowExplode,
mrc); NextPrep(dpNext, mrc); NextPrep(jrcNext, nrc); initialLoad
= TRUE;
nrSet  =  0;  nid  =  1;  RCSetTolerance();  DolClear();}  void
RCSetTolerance()
{iLaw = mrc*nrc*20; iLtw = mrc*nrc* 2; iLggFactorTolerance =
0.0003;
iLggCount = 16; long rwiter =LONG_MAX; long rwtimeSet =LONG_MAX;
SetrwMax(rwiter, rwtimeSet); prec rwSliceMin = 0.001f;
prec   rwSliceMax   =   0.250f;   prec   rwSliceFac   =   0.001f;
SetRWSlice(rwSliceMin,
rwSliceMax, rwSliceFac); int pairMinCt = 2; prec pairMinInc =
0.000001f;
int pairMaxCt = 10; prec blockMinInc = 0.000005f; int blockMaxCt
= 5;
prec  walkMinInc  =  0.00001f;  PairMan::SetTolerance(pairMinCt,
pairMinInc,
pairMaxCt, blockMinInc, blockMaxCt, walkMinInc);}
int GenrSetId(NEXTs& ieList) {int i; for(i=nrSet-1; 0<=i; i--)
if(NextEqual(rSetieNext[i],ieList)) return i; NextCopy(ieList,
rSetieNext[nrSet]);  return  nrSet++;}  int  GetrSetId()  {return
nrSet;}
void LoadGroup(NEXTs& ieList, int jrc, ResConduit& grc) {int ie;
int ig = ieList.lo; RCG.nir = grc.nir; ARRAYCOPY(grc.rstop [0],
RCG.rstop[0], RCG.nir+1); ARRAYCOPY(grc.estop [0],RCG.estop[0],
RCG.nir+1);
ARRAYCOPY(grc.dedr [0],RCG.dedr[0], RCG.nir);
ARRAYCOPY(grc.dedrInfinite[0],RCG.dedrInfinite[0], RCG.nir+1);
RCG.rSet     =     GenrSetId(ieList);     RCG.id     =     nid++;
NextInsert(headNextb[jrc],
ig); LOOPs(ieList,ie) {RCE.pFactorDep = &(RCG.factor);
RCE.ieDown = NextFollow(ieList,ie); RCE.igUp = ig; RCE.type =
rctNor;
RCE.rSet      =      RCG.rSet;      NextInsert(horNextb[ie],jrc);
NextInsert(irfNextb,ie)
; if(grc.IsConvex()) NextInsert(irowExplode, ie);}}
void LoadGroupDP(NEXTs& ieList, int jrc, prec slope /*=1.0*/)
{int ie;
int ig = ieList.lo; RCG.dirPut = TRUE; RCG.dedr[0] = slope;
RCG.rstop[1]
= bigM/slope; RCG.estop[1] = bigM; RCG.nir = 1;
RCG.rSet     =     GenrSetId(ieList);     RCG.id     =     nid++;
NextInsert(headNextb[jrc],
ig); LOOPs(ieList,ie) {RCE.pFactorDep = &(RCG.factor);
RCE.ieDown = NextFollow(ieList,ie); RCE.igUp = ig; RCE.type =
rctNor;
RCE.rSet      =      RCG.rSet;      NextInsert(horNextb[ie],jrc);
NextInsert(irfNextb,ie)
;   NextInsert(dpNext,ie);}}   void   LoadqOrg(prec*   pData)
{ARRAYCOPY(pData[0],
```

```
qOrg[0],     nrc);}     void     LoadPotential(prec*     pData)
{ARRAYCOPY(pData[0],
potential[0], mrc);} void SetrcTypeAsFix(int ig, int jrc, prec
allotment)
{RCG.allotment = allotment; GenGroupFactor(ig,jrc,FALSE);
RCG.type  =  rctFix;}  BOOL  IsGoodrcTypeAsFix()  {int  ig,  jrc;
for(jrc=0;
jrc<nrc;jrc++) {prec aquant = qOrg[jrc]; igbLOOP if(RCG.type ==
rctFix)
aquant  -=  RCG.allotment;  if(ZeroPress(aquant)  <  0)  return
FALSE;}
return  TRUE;}  rcType  GetrcType(int  ig,  int  jrc)  {return
RCG.type;}
void SetrwMax(long iter, prec timeSet) {iLrw = iter; iLrwTime =
timeSet;}
void SetRWSlice(prec sliceMin, prec sliceMax, prec sliceFac)
{rwSliceMin  =  sliceMin;  rwSliceMax  =  sliceMax;  rwSliceFac  =
sliceFac;}
void GetRWSlice(prec& sliceMin, prec& sliceMax, prec& sliceFac)
{sliceMin  =  rwSliceMin;  sliceMax  =  rwSliceMax;  sliceFac  =
rwSliceFac;}
prec GetqOrgwtIn(int jrc) {return qOrgwtIn[jrc];}
void  IncAllotmentTest(int  ig,  int  jrc,  prec  increment)  {int
ie,j;
if(!IsDolCol(jrc) || jrc == dolCol)
RCG.allotment += increment * qOrgwtIn[jrc];
else {increment *= dolPrice[jrc];
RCG.allotment     +=     increment     *     qOrgwtIn[jrc];}
GenGroupFactor(ig,jrc,FALSE);
ieLOOP {rowFactor[ie] = 1; LOOPs(horNextb[ie],j) rowFactor[ie]
*= rc[ie]
[j].factor;}} prec GetColSumAllotqOrgwtOt(int jrc)
{prec   colsum   =   headHCb[jrc].GetSumAllot();   colsum   *=
qOrgwtOt[jrc];
return colsum;} int GetColType(int jrc) {int k; ResConduit* prc;
prec  colAbsorb  =  0;  prcLOOPhc(headHCb[jrc])  if(prc->type  !=
rctFix)
colAbsorb  +=  prc->rstop[prc->nir];  else  colAbsorb  +=  prc-
>allotment;
if(IsEqualQ(colAbsorb,qOrg[jrc])) return 0; else if(qOrg[jrc] <
colAbsorb)
return  -1;  else  return  1;}  void  RFiNextEqualPurge(NEXTs&  ns)
{int ibase,
ifol; LOOPs(ns,ibase) {ifol = ibase + 1; RESUMELOOPs(ns,ifol)
if(NextEqual(horNextb[ibase],horNextb[ifol])) {BOOL sameGroup =
TRUE;
int  jrc;  LOOPs(horNextb[ibase],jrc)  if(rc[ibase][jrc].igUp  !=
rc[ifol]
[jrc].igUp)  sameGroup  =  FALSE;  if(ibase  <  rcmProd  &&  ifol  <
rcmProd)
for(jrc=0;jrc<nrc;jrc++)       if(dpTieb[ibase][jrc]       !=
dpTieb[ifol][jrc])
```

```
sameGroup = FALSE; else {int idp = dpTieb[ibase][jrc]; if(idp !=
-1)
if(!IsEqual(CTGetOrgaElement(idp,   ibase),CTGetOrgaElement(idp,
ifol)))
sameGroup  = FALSE;} if(sameGroup) NextDelete(ns,ifol);}}} void
SetRCType()
{int k; int jrc; ResConduit* prc; LOOPs(jrcNextHCb, jrc)
{if(GetColType(jrc)   ==   -1   &&   1   <   (headHCb[jrc].nele   -
headHCb[jrc]
.GetFixCt())) {prcLOOPhc(headHCb[jrc]) if(prc->type != rctFix)
prc->type = rctVar;} else {prcLOOPhc(headHCb[jrc]) if(prc->type
!= rctFix)
prc->type  =  rctBas;}}} void SetMarginCtlW(BOOL seekVar, BOOL
seeknonFix)
{RCFilter     rcf;     rcf.Include(rctVar);     if(seeknonFix)
rcf.Include(rctBas);
SetMarginCtlFinNew(rcf);} void SetMarginCtlB() {RCFilter rcf;
rcf.SetAll()
; SetMarginCtlFinNew(rcf);} void SetMarginCtlFinNew(RCFilter&
rcf) {int i,
j; int ig, ie, jrc; int ipull; int ctMarV[CORTMAX_M1]={0};
int ctMarH[RCDTMAX_NRC]={0}; for(jrc=0;jrc<nrc;jrc++)
igbLOOP    if(rcf.Pass(rc[ig][jrc]))    {ieLOOP    ctMarV[ie]++;
ctMarH[jrc]++;}
for(i=0;i<mrc;i++) NextClear(horNextw[i]); for(j=0;j<nrc;j++)
NextClear(headNextw[j]);                      NextClear(irfNextw);
NextClear(jrcNext);
for(jrc=0;jrc<nrc;jrc++) {igbLOOP if(rcf.Pass(rc[ig][jrc]))
{NextInsert(headNextw[jrc], ig); NextInsert(jrcNext, jrc);
ieLOOP    if(ie    !=    dolRow    ||    jrc    ==    dolCol)
{NextInsert(horNextw[ie],jrc);
NextInsert(irfNextw,ie);}}}                 SetMarginCtlFinOld();
NextPrep(jrcNextHCw,
nrc);  LOOPs(jrcNextHCb,  jrc)  {headHCw[jrc].Load(headHCb[jrc],
rcf);
if(headHCw[jrc].nele)       NextInsert(jrcNextHCw,       jrc);}
dolHCw.Load(dolHCb,
rcf); LOOPs(ipullNext,  ipull)  {dolHCw.InInit(dolRWPull[ipull],
FALSE);
dolHCw.InInit(dolRWDP  [ipull],  FALSE);  LOOPs(horNextw[ipull],
jrc)
if(IsDolCol(jrc))        {ig        =        rc[ipull][jrc].igUp;
dolHCw.In(dolRWPull[ipull],
ig, jrc); ig = dpTieb[ipull][jrc]; if(ig != -1 && RCG.type !=
rctFix)
dolHCw.In(dolRWDP[ipull], ig, jrc);} dolRWPullRev[ipull]
=        dolRWPull[ipull];        dolRWPullRev[ipull].Reverse();
dolRWDPRev[ipull]
= dolRWDP[ipull]; dolRWDPRev[ipull].Reverse(); dolHCdp [ipull]
.Load(dolHCw, &dolRWDP[ipull]);} int k; ResConduit* prc;
dolHCb.InInit(dolFix, FALSE); prcLOOPhc(dolHCw) if(prc->type ==
rctFix)
```

```
dolHCb.In(dolFix, prc); dolFixRev = dolFix; dolFixRev.Reverse();
TWSetMargins();} void SetMarginCtlFinOld() {int iProd, jrc;
PairMan::LoadigPrep();                                   jrcLOOP
PairMan::Loadig(jrc,headNextw[jrc]);
FFPBuild(); for(iProd=0; iProd<rcmProd; iProd++)
{BOOL  dolActinCol  =  FALSE;  NextClear(dpTieNext  [iProd]);
for(jrc=0;
jrc<nrc;jrc++) if(!IsDolCol(jrc)) {int ig = dpTieb[iProd][jrc];
if(ig     !=     -1     &&     horNextw[iProd].use[jrc]     &&
horNextw[ig].use[jrc])
NextInsert(dpTieNext[iProd],jrc);}
NextClear(dpTieNextDol[iProd]);
if(dolAct && dpTieb[iProd][dolCol] != -1) LOOPs(horNextw[iProd],
jrc)
if(IsDolCol(jrc)) NextInsert(dpTieNextDol[iProd], jrc);}
OrientPotentialxx();}    void    FFPBuild()    {int    i,    jrc;
for(i=0;i<mrc;i++)
{ffp[i].Init();                LOOPs(horNextb[i],            jrc)
ffp[i].NoteRC(rc[i][jrc],
horNextw[i].use[jrc]);}}  void  OrientPotentialxx()  {int  i;
iLOOPs(dpNext)
{potential[i]  =  1;}  for(i=0;i<mrc;i++)  {ffp[i].potential  =
potential[i];
GenRowFactor(i);} ARRAYCOPY(potential[0],potentialCTwt[0],mrc);
CTFactorInb(potentialCTwt);              ARRAYCOPY(potential[0],
lwOrgpotential [0],
mrc); ARRAYCOPY(potentialCTwt[0],lwOrgpotentialCTwt[0],mrc);
NextCopy(irfNextw,ipullNext);        NextCopy(irfNextw,ilwNext);
iLOOPs(irfNextw)
{if(!ZeroPress(potential[i])) NextDelete(ipullNext, i);
if(rcmProd <= i || !ZeroPress(potential[i])) NextDelete(ilwNext,
i);}
NextSub(ipullNext,     dpNext);     if(dolRow      !=      -1)
NextInsert(ipullNext, dolRow)
;    RFiNextEqualPurge(ipullNext);    iLOOPs(ipullNext)    {prec
OneminusTol = 1.0;
OneminusTol   -=   TOLERANCE;   maxRowFac[i]   =   OneminusTol;}
iLOOPs(dpNext)
maxRowFac[i] = prec_MAX;} void GenGroupFactor(int ig, int jrc,
BOOL updaterowFactor) {int ie, irw; prec factor; RCG.onCorner =
FALSE;
for(RCG.ir=0;RCG.ir<RCG.nir;RCG.ir++)
if(RCG.allotment < RCG.rstop[RCG.ir+1]) break; irw = RCG.ir;
if(IsEqualQ(RCG.allotment,RCG.rstop[RCG.ir])   ||   RCG.ir   ==
RCG.nir)
RCG.onCorner              =              TRUE;              else
if(IsEqualQ(RCG.allotment,RCG.rstop[RCG.ir+1]))
{RCG.ir++; RCG.onCorner = TRUE;} if(RCG.allotment < 0) {}
factor  =  RCG.estop[irw]  +  (RCG.allotment  -  RCG.rstop[irw])  *
RCG.dedr[irw]
; if(RCG.onCorner) if(RCG.ir) RCG.dedrs = RCG.dedr[RCG.ir-1];
else RCG.dedrs = gmv_MAX; else RCG.dedrs = RCG.dedr[RCG.ir];
```

```
RCG.dedra = RCG.dedr[RCG.ir]; ieLOOP {*(RCE.pFactorDep)
= RĈE.factor = factor; RCE.dedrs = RCG.dedrs; RCE.dedra =
RCG.dedra;
if(updaterowFactor) GenRowFactor(ie);}} void GenRowFactor(int i)
{rowFactor[i] = ffp[i].GetFactor_etal();} void GenRowFactor()
{int i;
iLOOPs(irfNextb) GenRowFactor(i);} void GenRowFactor(NEXTs&
rfNext)
{int i; iLOOPs(rfNext) GenRowFactor(i);} prec GetRowFactor(int
i)
{return rowFactor[i];} void GenGroupMV(int ig, int jrc)
{if(ig != dolRow || jrc == dolCol) {int ie; RCG.gmvs = 0;
RCG.gmva = 0;
ieLOOP {XctMVGet(ie); if(!RCE.subBlk) {RCE.emvs = ctvMVs[ie]
* RCE.dedrsBfPot; if(RCE.emvs < TOLERANCE) RCE.emvs = 0;}
else RCE.emvs = gmv_MAX; RCE.emva = ctvMVa[ie] * RCE.dedraBfPot;
if(RCE.emva < TOLERANCE) RCE.emva = 0; RCG.gmvs += RCE.emvs;
RCG.gmva += RCE.emva;} if(RCG.onCorner) {if(RCG.ir == 0)
RCG.gmvs = gmv_MAX; if(RCG.ir == RCG.nir) RCG.gmva = 0;} if(jrc
== dolCol)
{RCG.gmvs += 1; RCG.gmva += 1;}} else {if(RCG.onCorner && RCG.ir
== 0)
RCG.gmvs = gmv_MAX; else RCG.gmvs = 1;}} void GenNextMV(NEXTs&
ns)
{int ig, jrc; XctMVPre(); LOOPs(ns,jrc) igwLOOP
GenGroupMV(ig,jrc);
XctMVPost();} void GenColMV(int jrc) {int ig; XctMVPre();
igwLOOP GenGroupMV(ig,jrc); XctMVPost();} void GenPairMV(int
ig1, int ig2,
int jrc) {XctMVPre(); GenGroupMV(ig1,jrc); GenGroupMV(ig2,jrc);
XctMVPost();} void PermBlockPair(int igs, int iga, int jrcs, int
jrca)
{if(jrcs == jrca) {awpm.PermBlockPair(igs,iga,jrcs);
twpm.PermBlockPair(igs,iga,jrca);} if(dolHCw.Has(igs, jrcs)
&& dolHCw.Has(iga, jrca)) dolHCw.PairBlock(igs, iga, jrcs,
jrca);
permanentBlockPair = TRUE;} BOOL RCGetPermBlockPair(int igs, int
iga,
int jrc) {return awpm.GetPermBlockPair(igs, iga, jrc);}
void GtoGForceThru(int igs, int iga, int jrc, prec quant)
{GtoGForceThru(igs, iga, jrc, jrc, quant);} void
GtoGForceThru(int igs,
int iga, int jrcs, int jrca, prec quant) {IncAllotment(igs,
jrcs, -quant,
TRUE); IncAllotment(iga, jrca, quant, TRUE); LoadCTbOrg();
MakeCTFeasible();} void GtoGTransfer(int igs, int iga, int jrc,
BOOL callByLW/*=FALSE*/, BOOL callByRW/*=FALSE*/,
prec minFactor /*=0.0f*/, prec maxFactor /*=bigM*/)
{GtoGTransfer(igs,
jrc, iga, jrc, callByLW, callByRW, minFactor, maxFactor);}
void GtoGTransfer(int igs, int jrcs, int iga, int jrca, BOOL,
BOOL callByRW/*=FALSE*/, prec minFactor /*=0.0f*/,
```

```
prec maxFactor /*=bigM*/) {int ig, ie, jrc; int rtCond; prec
inFactor;
prec outFactor; prec temp; BOOL raDone = FALSE; BOOL refresh;
int nIteration = 0; prec bOrgBak[CORTMAX_M1];
NextClear(bIncNext);
ig = igs; jrc = jrcs; ieLOOP bOrgBak[ie] = CTGetbOrgBak(ie);
do {inFactor = maxFactor; ig = igs; jrc = jrcs;
if(GDecMax(ig,jrc)
< inFactor) inFactor = GDecMax(ig,jrc); if(ig != dolRow || jrc
== dolCol)
{if(RCG.dirPut) {ieLOOP NextInsert(bIncNext,ie, -potential[ie]
* RCG.dedrs);} else {ieLOOP {temp = RCE.dedrsBfPot;
NextInsert(bIncNext,
ie,-temp);}}} ig = iga; jrc = jrca; if(GIncMax(ig,jrc) <
inFactor)
inFactor = GIncMax(ig,jrc); if(ig != dolRow || jrc == dolCol)
{if(RCG.dirPut) ieLOOP NextInsert(bIncNext,ie, potential[ie] *
RCG.dedra);
else ieLOOP {temp = RCE.dedraBfPot;
NextInsert(bIncNext,ie,temp);}}
CTFactorInb(bIncNext); CTIncb(inFactor, bIncNext, outFactor,
rtCond);
refresh = FALSE; if(outFactor<=0) {RCRoundAdjustment(FALSE);
raDone = TRUE; refresh = TRUE; if(!callByRW) {if(RCShc.gmvs <
RCAhc.gmva)
{CTIncb(inFactor, bIncNext, outFactor, rtCond); if(outFactor<=0)
{PermBlockPair(igs, iga, jrcs, jrca); return;}} else {return;}}
else {CTIncb(inFactor, bIncNext, outFactor, rtCond);
if(outFactor <= 0)
outFactor = inFactor;}} IncAllotment(igs, jrcs, -outFactor,
TRUE);
IncAllotment(iga, jrca, outFactor, TRUE);
if(jrcs == jrca && outFactor < iLggColumnTolerance[jrcs])
if(iLpm.IncPairCount(igs, iga, jrcs) == iLggCount)
PermBlockPair(igs, iga,
jrcs, jrca); ig = igs; jrc = jrcs;
ieLOOP {static const prec tol2 = 0.00001f; prec org =
CTGetbOrg(ie);
prec rowpot = rowFactor[ie] * potential[ie]; if((!
ZeroPress(rowFactor[ie]
) && !(ZeroPress(bOrgBak[ie])) && ZeroPress(CTGetb(ie)))
|| !IsEqualComP(org, rowpot, tol2)) {refresh = TRUE; break;}}
if(refresh)
{RCRoundAdjustment(FALSE);} if(callByRW) {if(outFactor <
minFactor)
{IncAllotment(igs, jrcs, outFactor, FALSE); IncAllotment(iga,
jrca,
-outFactor, FALSE); if(inFactor < minFactor) {minFactor =
__min(minFactor,
RCShc.allotment); minFactor = __min(minFactor,
(RCAhc.rstop[RCAhc.nir]
```

```
- RCAhc.allotment));} GtoGForceThru(igs, iga, jrcs, jrca,
minFactor);}
return;} else {GenGroupMV(igs, jrcs); GenGroupMV(iga, jrca);
if(RCAhc.gmvs < RCShc.gmva) {if(!raDone) {IncAllotment(igs,
jrcs,
outFactor, FALSE); IncAllotment(iga, jrca, -outFactor, FALSE);
RCRoundAdjustment(FALSE); raDone = TRUE; if(TRUE)
{if(RCShc.gmvs < RCAhc.gmva) {}
else if(RCAhc.gmvs < RCShc.gmva && gtogTransferOKSwapDirection)
{Swap(igs,
iga); Swap(jrcs, jrca); continue;} else {return;}}} else {int
nwhile = 0;
prec shaveBack = outFactor; while(RCAhc.gmvs < RCShc.gmva)
{shaveBack *= 0.5f; if(shaveBack < 1e-10) {PermBlockPair(igs,
iga, jrcs,
jrca); return;} IncAllotment(igs, jrcs, shaveBack, TRUE);
IncAllotment(iga, jrca, -shaveBack, TRUE); LoadCTbOrg();
MakeCTFeasible();
GenGroupMV(igs, jrcs); GenGroupMV(iga, jrca);}}} else {raDone =
FALSE;}}
if(++nIteration >= 40) {PermBlockPair(igs, iga, jrcs, jrca);
return;}}
while(RCShc.gmvs < RCAhc.gmva);} int GetRowMinS(int jrc) {int
ig, mini=-1;
prec minv = gmv_MAX; igwLOOP if(RCG.gmvs < minv) {minv =
RCG.gmvs;
mini = ig;} return mini;} int GetRowMinS(int igbut, int jrc)
{int ig,
mini=-1; prec minv = gmv_MAX; igwLOOP if(RCG.gmvs < minv && ig
!= igbut)
{minv = RCG.gmvs; mini = ig;} return mini;} prec GIncMax(int ig,
int jrc)
{if(RCG.ir == RCG.nir) return 0; else return RCG.rstop[RCG.ir+1]
- RCG.allotment;} prec GDecMax(int ig, int jrc)
{if(RCG.onCorner)
if(RCG.ir) return RCG.allotment - RCG.rstop[RCG.ir-1]; else
return 0;
else return RCG.allotment - RCG.rstop[RCG.ir];} void
IncAllotment(int ig,
int jrc, prec increment, BOOL updaterowFac) {RCG.allotment +=
increment;
GenGroupFactor(ig,jrc,updaterowFac);} void XctMVPre() {}
void XctMVGet(int i) {if(!ctvMVcur[i]) {CTGetvData(i);}} void
XctMVPost()
{} void LoadCTbOrg() {prec rcbOrg[CORTMAX_M1]; int i;
for(i=0;i<mrc;i++)
rcbOrg[i] = rowFactor[i] * potentialCTwt[i]; CTLoadb(rcbOrg);}
void RCRoundAdjustment(BOOL doWalk/*=TRUE*/)
{rcRoundAdjustmentCt++;
int jrc; LOOPs(jrcNextHCw, jrc) if(roundAdjustmentOK[jrc]) {HCol
tHC;
RCFilter rcf; rcf.Include(rctFix); rcf.Include(rctBas);
```

```
prec qtFix = headHCb[jrc].GetSumAllot(rcf); headHCw[jrc]
.RoundAdjustment(qOrg[jrc] - qtFix, FALSE);} GenRowFactor();
LoadCTbOrg();
CTRoundAdjustment(); GenNextMV(jrcNext); if(doWalk) Walk();}
void RCRoundAdjustmentwAW() {RCRoundAdjustment(FALSE); int
rtCond;
AxisWalk(rtCond);} void MakeCTFeasible() {if(CTMakeFeasible() ==
1)
RCRoundAdjustment(FALSE);} void IncCTb(prec facIn, NEXTs& bSupl,
prec& facOut) {int rtCond; CTFactorInb(bSupl); CTIncb(facIn,
bSupl,
facOut, rtCond); if(rtCond == 1) {RCRoundAdjustment(FALSE);
facOut = 0;}}
void LoadCTb_bOrg(prec* inb, prec* inbOrg, BOOL negb) {int
rtCond;
rtCond = CTbPlayLoadbbOrg(inb, inbOrg, negb); if(rtCond==1)
CTRoundAdjustment();} void BPPrep() {NextPrep(bpiNext, mrc);
NextPrep(bpiTouchNext, mrc);} void BPreadybase() {int i,j;
for(i=0;i<mrc;
i++) bpbase.bOrg[i] = CTGetbOrgWT(i); iLOOPs(bpiNext)
{bpbase.bOrg[i] = 1;
jLOOPs(horNextb[i]) if(!bpFacVar[i][j]) bpbase.bOrg[i] *=
rc[i][j].factor;
}} int BPGenadjust(int i, prec bele) {prec lb, ub;
if(CTIsZerob(i))
{lb = -(TOLERANCE + bpTipOver); ub = - TOLERANCE/3.0f;}
else {lb = -bpTipOver; ub = 0;} if(bele < lb) return -1;
else if(ub < bele) return 1; else return 0;} int BPMinb(BPds&
bp)
{int mini; ARRAYMIN(bp.b, mini, mrc, bp.minv); bp.negb =
(bp.minv < 0);
bp.adjust = BPGenadjust(mini, bp.minv); return mini;}
void BP_BbyVec(BPds& bp) {CTbPlayBbyVec(bp.bOrg,bp.b);
BPMinb(bp);}
void BPprepSmartBbybpcur() {NextReverse(bpiNext,bpirNext);
ZEROOUT(bpbase.b[0],mrc);
CTbPlayBbyVec(bpirNext,bpbase.bOrg,bpbase.b);
CTbPlayPrepMaxNegb();} void BPSmartBbybpcur(BOOL reset,
BOOL acceptPosAdjust) {static NEXTs seqNext; static int count;
ARRAYCOPY(bpbase.b[0],bpcur.b[0],mrc); if(reset || !(--count))
{int mini;
CTbPlayBbyVec(bpiNext,bpcur.bOrg,bpcur.b); mini = BPMinb(bpcur);
if(acceptPosAdjust && bpcur.adjust==1) bpcur.adjust = 0;
if(bpcur.adjust)
{CTbPlayMaxNegb(bpbase.b,bpcur.b,mini,bpTipOver);
bpcur.minv = bpcur.b[mini]; bpcur.negb = (bpcur.minv < 0);
bpcur.adjust = BPGenadjust(mini, bpcur.minv);
if(acceptPosAdjust && bpcur.adjust==1) bpcur.adjust = 0;
if(bpcur.adjust)
{NextFill(seqNext, mrc); ARRAYCOPY(bpcur.b[0], seqNext.val[0],
mrc);
```

```
NextSortIV(seqNext,TRUE);   count   =   25;}}}   else   {int   i;
iLOOPs(seqNext)
{CTbPlayBbyVec(bpiNext,     bpcur.bOrg,     i,     bpcur.b[i]);
if(BPGenadjust(i,
bpcur.b[i]) == -1) {bpcur.minv = bpcur.b[i]; bpcur.negb = TRUE;
bpcur.adjust  =  -1;  return;}}  BPMinb(bpcur);}}  void  BPlay(void
(genb)
(BPds  &bp,  prec  e),  prec mine,  prec maxe)  {if(!(mine  <  maxe))
{genb(bpcur,
maxe);    BP_BbyVec(bpcur);    return;}    BPprepSmartBbybpcur();
genb(bpcur,maxe);
BPSmartBbybpcur(TRUE, TRUE); if(bpcur.adjust) {genb(bplo,0);
BP_BbyVec(bplo);        if(bplo.negb)        {CTRoundAdjustment();
BPprepSmartBbybpcur()
;  BPSmartBbybpcur(TRUE,TRUE);  BP_BbyVec(bplo);  if(bplo.minv  <
0.0)
bplo.minv    =    0.0;    if(!bpcur.adjust)    genb(bpcur,maxe);}
if(bpcur.adjust)
{prec e, laste; int bpParaCt; bplo.e = mine; bphi.e = bpcur.e;
bphi.minv   =   bpcur.minv;   e   =   Interpolate(bplo.minv,bplo.e,
bphi.minv,
bphi.e,  -  bpTipOver*0.5);  if(bplo.e  <  e  &&  e  <  bphi.e)
{bpParaCt=0;}
else {e = (bplo.e + bphi.e)*0.5; bpParaCt = bpParaMax;}
if(!(bplo.e   <   e   &&   e   <   bphi.e))   {genb(bpcur,maxe);
BP_BbyVec(bpcur);
return;} genb(bpcur,e); BPSmartBbybpcur(FALSE, FALSE); laste = -
1;
while(bpcur.adjust  &&  laste  !=  e)  {laste  =  e;  if(bpParaCt  <
bpParaMax)
{e    =    Interpolate(bplo.minv,bplo.e,    bphi.minv,bphi.e,
bpcur.minv,bpcur.e,
- bpTipOver*0.5); if(bplo.e < e && e < bphi.e) bpParaCt++;
else  bpParaCt  =  bpParaMax;}  if(bpcur.adjust  <  0)  {bphi.e  =
bpcur.e;
bphi.minv  =  bpcur.minv;}  else  {bplo.e  =  bpcur.e;  bplo.minv  =
bpcur.minv;}
if(bpParaMax   ==   bpParaCt)   e   =   (bplo.e   +   bphi.e)*0.5;
genb(bpcur,e);
BPSmartBbybpcur(FALSE,     FALSE);}}}}     void     BPFin(BOOL
genRowFactor/*=TRUE*/)
{int   i;   iLOOPs(bpiNext)   {if(genRowFactor)   GenRowFactor(i);
bpbase.bOrg[i]
= rowFactor[i] * potentialCTwt[i];} BP_BbyVec(bpbase);
LoadCTb_bOrg(bpbase.b, bpbase.bOrg, bpbase.negb);} void AWPrep()
{NextPrep(bIncNext, mrc); NextPrep(jrcawNext,nrc);}
void AxisWalk(int& rtCond) {long awCtDw = iLaw; BOOL didGtoG,
dwProfitable;    awpm.WalkInit();    awpm.Clear(jrcNext);    do
{NextSub(jrcNext,
jrcNextDol); do {int ig, igs, iga, jrc; awpm.ItPrep();
jrcLOOP {awpm.ItPrepCol(jrc); igwLOOP {awpm.ItNoteSub(RCG.gmvs);
awpm.ItNoteAdd(RCG.gmva);}} awpm.GetBestPair(igs,iga,jrc);
```

```
didGtoG = FALSE; while(igs != -1) {didGtoG = TRUE;
GtoGTransfer(igs, iga,
jrc); awCtDw--; awpm.ItUpdateValueSub(igs, jrc, RCS.gmvs);
awpm.ItUpdateValueAdd(igs, jrc, -1); awpm.ItUpdateValueAdd(iga,
jrc,
RCA.gmva); awpm.ItUpdateValueSub(iga, jrc, bigM);
awpm.GetBestPair(igs,
iga,jrc); if(igs != -1) {GenPairMV(igs,iga,jrc); if(RCA.gmva <=
RCS.gmvs)
igs = -1;}} if(didGtoG) GenNextMV(jrcNext);} while(didGtoG && 0
<=awCtDw);
NextAdd(jrcNext, jrcNextDol); DolWalk(dwProfitable, awCtDw);}
while(dwProfitable && 0 <=awCtDw); GenNextMV(jrcNext); rtCond =
0;}
void ExplodeWalk(int& rtCond) {int i, rtCond2;
rwbestProfit = CTGetProfit(); rwbestImage.Out(); rwProfitable =
FALSE;
LOOPs(irowExplode, i) {prec potentialHold = 0; prec
potentialCTwtHold = 0;
Swap(potential [i], potentialHold); Swap(potentialCTwt[i],
potentialCTwtHold); ffp[i].potential = potential[i];
GenRowFactor(i);
LoadCTbOrg(); CTMakeFeasible(); GenNextMV(jrcNext);
AxisWalk(rtCond2);
Swap(potential [i], potentialHold); Swap(potentialCTwt[i],
potentialCTwtHold); ffp[i].potential = potential[i];
GenRowFactor(i);
LoadCTbOrg(); CTMakeFeasible(); if(rwbestProfit < CTGetProfit())
{rwProfitable = TRUE; rwbestProfit = CTGetProfit();
rwbestImage.Out();}
else {rwbestImage.In();}} GenNextMV(jrcNext); if(rwProfitable)
ATLManager(rtCond2); rtCond = 0;} void TWPrep() {int i,j; int
jrc;
NextPrep(itopNext, mrc); NextPrep(jrctwNext,nrc);
for(jrc=0;jrc<nrc;jrc++)
NextPrep(igtwNext[jrc],mrc); for(i=0;i<mrc;i++)
NextPrep(rowrSetNext[i],
nrSet); NextPrep(twrSetNext,nrSet); for(j=0;j<nrSet;j++)
NextPrep(pendrSetNext[j],mrc); NextPrep(twGenFactorNext,mrc);}
void TWPrepWalkSession() {} void TWSetMargins() {int ie, ig,
jrc;
for(ie=0;ie<mrc;ie++) {NextClear(rowrSetNext[ie]);
LOOPs(horNextw[ie],jrc)
{ig = RCE.igUp; NextInsert(rowrSetNext[ie],RCG.rSet);}}}
BOOL TWisTopped(int iRow) {if(ctvMVa[iRow]
< TOLERANCE && TOLERANCE < ctvMVs [iRow]
&& rowFactorTolerance < rowFactor [iRow]
&& TOLERANCE < potentialCTwt[iRow]) {return TRUE;} else return
FALSE;}
void TWGenitopNext(int lwControl) {int i, j; int ie, ig, jrc;
NextClear(itopNext); XctMVPre(); iLOOPs(irfNextw) {XctMVGet(i);
if(TWisTopped(i) && 1 < horNextw[i].nele) {int j, ctMultiRC = 0;
```

```
jLOOPs(horNextw[i])     if(1<headNextw[j].nele)     ctMultiRC++;
if(1<ctMultiRC)
NextInsert(itopNext,i);}} XctMVPost();
if(0<=rwiHAT && rowFactorTolerance < rowFactor [rwiHAT]
&&  TOLERANCE   <   potentialCTwt[rwiHAT]   &&   1   <   horNextw
[rwiHAT].nele)
{LOOPs(gOverLapRowNext[rwiHAT], ie) NextDelete(itopNext, ie);
NextInsert(itopNext,rwiHAT);}        RFiNextEqualPurge(itopNext);
if(!lwControl)
NextClear(jrctwNext); else {NextCopy(jrcNext, jrctwNext);
LOOPs_del(jrctwNext, jrc) if(!(1<headNextw[jrc].nele))
NextDelete(jrctwNext, jrc);} for(jrc=0;jrc<nrc;jrc++)
NextClear(igtwNext[jrc]);                   NextClear(twrSetNext);
for(j=0;j<nrSet;j++)
NextClear(pendrSetNext[j]);          LOOPs(itopNext,          ie)
LOOPs(horNextw[ie],jrc)
if(1<headNextw[jrc].nele)         {ig          =          RCE.igUp;
NextInsert(jrctwNext,jrc);
NextInsert(igtwNext[jrc],ig); NextInsert(twrSetNext, RCG.rSet);
NextInsert(pendrSetNext[RCG.rSet],ie);}}
prec TWGenNetFactor(ResConduit& rc, prec& qbase)
{prec   netFactor   =   rc.factor;   prec   difallot   =   qbase   -
rc.allotment;
if(0<difallot) netFactor+= rc.dedra * difallot;
else  {netFactor+=  rc.dedrs  *  difallot;  if(netFactor  <  0)
netFactor = 0;}
return netFactor;} void TWNotePossiblyBetterO(int ig, int jrc)
{if(minig[jrc].NoteImprovement(ig, RCG.twgmvs)) {int oig = ig;
igtwLOOP  {if(RCG.ir  !=  RCG.nir  &&  ig  !=  oig)  {int  mini  =
minig[jrc]
.GetBestBut(ig);    if(mini    !=    -1)    {prec    cost    =
(RCG.factor/RCG.dedra)
*                                        rc[mini][jrc].twgmvs;
if(rSetminj[RCG.rSet].NoteImprovement(jrc,cost))
{int  i;  iLOOPs(pendrSetNext[RCG.rSet])  pendrSet[i][RCG.rSet]  =
TRUE;
RCG.twuDownig  =  mini;}}}}}} void TWGetu(Pairint& v, int& uig,
int& ujrc)
{int  vig,  vjrc;  v.Get(vig,vjrc);  uig  =  RCV.twuig;  ujrc  =
RCV.twujrc;}
void     TWIncPairintv(Pairint&     v)     {int     uig,     ujrc;
TWGetu(v,uig,ujrc);
v.Load(RCU.twuDownig,ujrc);}
#define DEFabrs long double a, b, r, s; a = TWGenNetFactor(urc,
uqbase);\
b = urc.dedra; r = TWGenNetFactor(vrc, vqbase); s = vrc.dedrs;
prec TWufv(ResConduit& urc, prec uqbase, ResConduit& vrc, prec
vqbase)
{DEFabrs;  long  double  num  =  a*s;  long  double  dem  =  b*r;
if(Divisible(num,
dem) && 0 <= (num/dem)) return (prec) (num/dem); else return
bigM;}
```

```
prec TWufv(ResConduit& urc, prec uqbase, ResConduit& vrc, prec
vqbase,
prec vquant) {DEFabrs; long double sv = s * vquant;
long double num = a * sv; long double dem = b * (r - sv);
if(Divisible(num, dem) && 0 <= (num/dem)) return (prec)
(num/dem);
else return bigM;} prec TWvfu(ResConduit& urc, prec uqbase,
ResConduit& vrc, prec vqbase) {DEFabrs; long double dem = a*s;
long double num = b*r; if(Divisible(num, dem) && 0 <= (num/dem))
return (prec) (num/dem); else return 0;} prec TWvfu(ResConduit&
urc,
prec uqbase, ResConduit& vrc, prec vqbase, prec uquant)
{DEFabrs;
long double bu = b * uquant; long double dem = s * (a + bu);
long double num = r * bu; if(Divisible(num, dem) && 0 <=
(num/dem))
return (prec) (num/dem); else return 0;}
#undef DEFabrs
#undef Divide
void TWGenMVs() {int i; int vjrcMaster; int vig, vjrc; int uig,
ujrc;
int ig, ie, jrc; int urSet; int itop; BOOL changed; Pairint
vOrgOrg;
Pairint vOrgCycle; Pairint vOrg; Pairint vCur; twFootPrt = 0;
jrctwLOOP minig[jrc].Init(); jrctwLOOP igwLOOP {RCG.twuig = -1;
RCG.twFootPrt = 0; RCG.twgmvs = RCG.gmvs;} iLOOPs(twrSetNext)
rSetminj[i]
.Init(bigM/2.0f); jrctwLOOP igwLOOP TWNotePossiblyBetterO(ig,
jrc);
LOOPs(itopNext,itop) {NextCopy(horNextw[itop],
twhorNextv[itop]);
LOOPs(horNextw[itop], jrc) {ig = rc[itop][jrc].igUp;
if(!RCG.ir && RCG.onCorner) NextDelete(twhorNextv[itop],jrc);}}
do {changed = FALSE; LOOPs(itopNext,itop)
{LOOPs(rowrSetNext[itop],urSet)
{if(pendrSet[itop][urSet]) {pendrSet[itop][urSet] = FALSE;
LOOPs(twhorNextv[itop], vjrcMaster) {vjrc = vjrcMaster; vig =
rc[itop]
[vjrc].igUp; ujrc = rSetminj[urSet].GetBestBut(vjrc); if(ujrc !=
-1)
{uig = rSetieNext[urSet].lo; prec lkquant = TWufv(RCU,
RCU.allotment, RCV,
RCV.allotment); prec cost = RCuDown.twgmvs * lkquant;
LOOPs((*prSetSubNext[RCV.rSet][RCU.rSet]), ie) cost +=
RCVe.emvs;
LOOPs((*prSetSubNext[RCU.rSet][RCV.rSet]), ie)
cost -= RCUe.emva * lkquant; if(cost < 0) cost = 0;
if(cost + TOLERANCE < RCV.twgmvs) {changed = TRUE;
if(RCV.twuig != uig || RCV.twujrc != ujrc) {RCV.twuig = uig;
RCV.twujrc = ujrc; RCV.twgmvs = cost;
TWNotePossiblyBetterO(vig,vjrc);}
else {RCV.twgmvs = cost; TWNotePossiblyBetterO(vig,vjrc);
```

```
BOOL cycleEntertain; vOrgOrg.Load(vig, vjrc); twFootPrt++; vCur
= vOrgOrg;
while(TRUE) {vCur.Get(vig,vjrc); if(RCV.twuig == -1 ||
!RCV.twgmvs)
{cycleEntertain = FALSE; break;} else if(RCV.twFootPrt ==
twFootPrt)
{vOrgCycle = vCur; cycleEntertain = TRUE; break;}
else {RCV.twFootPrt = twFootPrt; TWIncPairintv(vCur);}}
if(cycleEntertain)
{vOrg = vOrgCycle; do {prec quant = 1; prec costInf = 0; vCur =
vOrg;
do {vCur.Get(vig,vjrc); TWGetu(vCur, uig, ujrc);
LOOPs((*prSetSubNext[RCV.rSet][RCU.rSet]), ie)
costInf += RCVe.emvs * quant; quant *= TWufv(RCU, RCU.allotment,
RCV,
RCV.allotment); LOOPs((*prSetSubNext[RCU.rSet][RCV.rSet]), ie)
costInf -= RCUe.emva * quant; vCur.Load(RCU.twuDownig,ujrc);}
while(vCur != vOrg); prec netQuant = 1.0f - quant; vOrg.Get(vig,
vjrc);
if(costInf < 0) {costInf = 0;} else {if(netQuant <= 0)
{TWIncPairintv(vOrg); continue;} else {costInf /= netQuant;}}
if(costInf <= RCV.twgmvs) RCV.twgmvs = costInf;
TWIncPairintv(vOrg);}
while(vOrg != vOrgCycle); if(FALSE) {vCur = vOrgCycle; do
{vCur.Get(vig,
vjrc); TWNotePossiblyBetterO(vig,vjrc); TWIncPairintv(vCur);}
while(vCur != vOrgCycle);} else {vCur = vOrgCycle; twFootPrt++;
vCur.Get(vig,vjrc); while(vig != -1 && RCV.twFootPrt !=
twFootPrt)
{RCV.twFootPrt = twFootPrt; TWNotePossiblyBetterO(vig,vjrc);
TWIncPairintv(vCur); vCur.Get(vig,vjrc);}}}}}}}}}}
while(changed);}
void TWGenMVa() {} void TWPushThru(prec snqt) {int i; int uig,
ujrc;
int vig, vjrc; prec quant = snqt; for(i=twnLink-1; 0<=i; i--)
{RCof(rcs[i]
).twQuant = RCof(rcs[i]).allotment - quant; if(i == iSplitVer)
{quant -= snqt; if(quant < 0) {quant = 0;}} RCof(rca[i])
.twQuant = RCof(rca[i]).allotment + quant; if(i == iSplitHor)
{prec debt = TWufv(RCsph,RCsph.allotment, vEnd, vEnd.allotment,
snqt);
quant -= debt; if(quant < 0) {quant = 0;} rca[i].Get(uig,ujrc);
rcs[i-1]
.Get(vig,vjrc); quant = TWvfu(RCU,RCU.allotment+debt, RCV,
RCV.allotment,
quant);} else if(i) {rca[i].Get(uig,ujrc); rcs[i-
1].Get(vig,vjrc);
quant = TWvfu(RCU,RCU.allotment, RCV, RCV.allotment, quant);}}}
void TWGenFactor(BPds& bbs, BOOL noteanchorfactorBut/* =FALSE
*/) {int i;
int ig, ie, jrc;
int iwatch = noteanchorfactorBut ? nstrayEanchorStart : -1;
```

```
ARRAYCOPY(bpbase.bOrg[0],bbs.bOrg[0],mrc);
for(i=0;i<nstrayG;i++)
{strayG[i].Get(ig,jrc);            RCG.twFactor              =
TWGenNetFactor(RCG,RCG.twQuant);}
for(i=0;i<nstrayE;i++) {strayE[i].Get(ie, jrc); ig = RCE.igUp;
RCE.twFactor  =  RCG.twFactor;  bbs.bOrg[ie]  *=  RCE.twFactor;
if(i+1==iwatch)
{rcs[twnLink-1].Get(ig,jrc);    ieLOOP   anchorfactorBut[ie]   =
bbs.bOrg[ie];}}}
int TWGentwMQuant() {int i; int uig, ujrc; int vig, vjrc;
prec quant = 1.0; for(i=twnLink-1; 0<=i; i--) {RCof(rcs[i])
.twMQuant = - quant; if(i == iSplitVer) {quant -= 1.0;
if(quant < TOLERANCE) return 1;} if(i == iSplitHor)
{if(!ZeroPress(vEnd.twQuant)) return 1; prec debt = TWufv(RCsph,
RCsph.twQuant, vEnd, vEnd.twQuant); if(quant < debt + TOLERANCE)
return 1;
RCof(rca[i]).twMQuant + debt; quant -= debt;} RCof(rca[i])
.twMQuant  =  +  quant;  if(i)  {rca[i].Get(uig,ujrc);  rcs[i-
1].Get(vig,vjrc);
quant *= TWvfu(RCU,RCU.twQuant, RCV, RCV.twQuant);}} return 0;}
void    TWevalMV(BPds&   bbs)   {int   i;   int   ig,   ie,   jrc;
TWGenFactor(bbs,TRUE);
if(TWGentwMQuant()==1) {bbs.netMV = -bigM; return;} prec netMV =
0;
for(i=0;i<nstrayE;i++) {strayE[i].Get(ie, jrc); ig = RCE.igUp;
if(nonLkct[ie]==1)  {if(strayE[i].bol)  netMV  +=  RCE.emva   *
RCG.twMQuant;
else netMV += RCE.emvs * RCG.twMQuant;} else {prec factorBut;
if(i<nstrayEanchorStart)                 {factorBut              =
bbs.bOrg[ie]/RCG.twFactor;}
else factorBut = anchorfactorBut[ie]; if(strayE[i].bol)
netMV += factorBut * (RCG.dedra * RCG.twMQuant) * (ctvMVa[ie]
*  potential[ie]);  else  netMV  +=  factorBut  *  (RCG.dedrs  *
RCG.twMQuant)
* (ctvMVs[ie] * potential[ie]);}} bbs.netMV = netMV;}
void TWbpFill(BPds& bbs, prec snqt) {int i; TWPushThru(snqt);
TWGenFactor(bbs);       iLOOPs(bpiNext)      bbs.bOrg[i]       *=
potentialCTwt[i];
bbs.e  =  snqt;}  void  TWSwapQin()  {int  i;  int  ig,  jrc;
for(i=0;i<twnLink;
i++) {rca[i].Get(ig,jrc); Swap(RCG.allotment, RCG.twQuant);
GenGroupFactor(ig,jrc,    FALSE);    if(i    !=    iSplitVer)
{rcs[i].Get(ig,jrc);
Swap(RCG.allotment,     RCG.twQuant);     GenGroupFactor(ig,jrc,
FALSE);}}
GenRowFactor(twGenFactorNext);} BOOL TWCrossHat(Pairint& p)
{if(rwiHAT != -1 && rc[p.i][p.j].igUp == rc[rwiHAT][p.j]
.igUp  &&  horNextw[rwiHAT].use[p.j])  return  TRUE;  else  return
FALSE;}
void TWLoadrcarcs(int iga, int igs, int jrc) {int i; int uig,
ujrc;
```

```
int vig, vjrc; BOOL crossOver = FALSE; rca[0].Load(iga,jrc);
rcs[0]
.Load(igs,jrc); twnLink = 1; iSplitVer = -1; iSplitHor = -1;
twcase = simplelink;
NextCopy((*prSetUnionNext[rc[igs][jrc].rSet][rc[iga]
[jrc].rSet]), twGenFactorNext); BOOL cont = TRUE; twFootPrt++;
RCS.twFootPrt = RCA.twFootPrt = twFootPrt; rcs[0].Get(vig,vjrc);
while(RCV.twuig != -1) {rca[twnLink].Load(RCV.twuig,
RCV.twujrc);
rca[twnLink].Get(uig,ujrc); rcs[twnLink].Load(RCU.twuDownig,
ujrc);
rcs[twnLink].Get(vig,vjrc); twnLink++; NextAdd(twGenFactorNext,
(*prSetUnionNext[RCU.rSet][RCV.rSet]));
if(RCU.twFootPrt == twFootPrt || RCV.twFootPrt == twFootPrt)
{crossOver = TRUE; break;}
else {RCU.twFootPrt = RCV.twFootPrt = twFootPrt;}} if(crossOver)
for(i=0;
i<twnLink-1; i++) {if(rca[i] == rca[twnLink-1]) {twcase =
splitnc;
twnLink--; break;} if(rcs[i] == rcs[twnLink-1]) {iSplitVer = i;
twcase = splitcyVer; break;} if(rca[i] == rcs[twnLink-1])
{if(rcs[i]
!= rca[twnLink-1]) {rcs[twnLink-1] = rcs[i]; iSplitVer = i;
twcase = splitcyVer;} else {twnLink--; twcase = splitnc;}
break;}
if(rcs[i] == rca[twnLink-1]) {twnLink--; twcase = splitnc;
break;}}
for(i=0;i<twnLink -1;i++) if(TWCrossHat(rcs[i]))
{if(!TWCrossHat(rca[i+1])
) {twcond = abortPair; return;}} else if(TWCrossHat(rca[i+1]))
{twcond = abortPair; return;}
if(iSplitVer == -1 && TWCrossHat(rcs[twnLink-1]))
{for(i=0;i<twnLink;i++)
if(TWCrossHat(rcs[i])) if(iSplitHor == -1) iSplitHor = i + 1;
else {twnLink = i + 1; twcase = splitcyHor; return;} twcond =
abortPair;}
return;} void TWCheckCleanCycle() {if(RCof(rca[0])
.rSet == RCof(rcs[twnLink - 1])
.rSet || twcase == splitcyVer || (twcase == splitcyHor &&
vEnd.rSet == RCsph.rSet)) {int iLink; int uig, ujrc; int vig,
vjrc; if(dolAct) {cleanCycle = FALSE; for(iLink=0; iLink <
twnLink; iLink++) {rca[iLink].Get(uig, ujrc); if(uig == dolRow)
return; rcs[iLink].Get(vig, vjrc); if(vig == dolRow) return;}}
cleanCycle = TRUE; for(iLink=0; iLink < twnLink - 1; iLink++)
{rca[iLink+1].Get(uig, ujrc); rcs[iLink].Get(vig, vjrc);
if(RCU.rSet != RCV.rSet) {cleanCycle = FALSE; return;}}} else
cleanCycle = FALSE;} void TWQuantSufficient() {if(twcond ==
normalTWCond && (IsEqualQ(RCof(rca[0]).twQuant,
RCof(rca[0]).allotment) || IsEqualQ(RCof(rcs[twnLink-
1]).twQuant, RCof(rcs[twnLink-1]).allotment))) twcond =
abortChoke;} void TWLoadbpiNext() {int ig, ie, jrc; int iLink;
int uig, ujrc; int vig, vjrc; NextClear(bpiNext);
```

```
rca[0].Get(ig,jrc); ieLOOP NextInsert(bpiNext,ie); for(iLink=0; iLink < twnLink - 1; iLink++) {rca[iLink+1].Get(uig, ujrc); rcs[iLink].Get(vig, vjrc); int urSet = RCU.rSet; int vrSet = RCV.rSet; if(urSet != vrSet) {LOOPs((*prSetSubNext[urSet][vrSet]),ie) NextInsert(bpiNext,ie); LOOPs((*prSetSubNext[vrSet][urSet]),ie) NextInsert(bpiNext,ie);}} rcs[twnLink-1].Get(ig,jrc); ieLOOP NextInsert(bpiNext,ie);} void TWLoadBPlayEquateMvMc() {int i; int ig, ie, jrc; int iLink; int uig, ujrc; int vig, vjrc; prec qtlo, qthi; ZEROOUT(nonLkct[0],mrc); NextClear(bpiNext); nstrayG = nstrayE = 0; strayG[nstrayG++] = rca[0]; rca[0].Get(ig,jrc); ieLOOP {NextInsert(bpiNext,ie); nonLkct[ie]++; strayE[nstrayE].Load(ie,jrc); strayE[nstrayE].bol = TRUE; nstrayE++;} for(iLink=0; iLink < twnLink - 1; iLink++) {rca[iLink+1].Get(uig, ujrc); rcs[iLink].Get(vig, vjrc); int urSet = RCU.rSet; int vrSet = RCV.rSet; if(iLink == iSplitVer) {strayG[nstrayG++].Load(uig,ujrc); LOOPs(rSetieNext[urSet],ie) {NextInsert(bpiNext,ie); nonLkct[ie]++; strayE[nstrayE].Load(ie,ujrc); strayE[nstrayE].bol = TRUE; nstrayE++;}} else if(iLink != iSplitHor -1) {if(urSet != vrSet) {if(prSetSubNext[urSet][vrSet]->nele) {strayG[nstrayG++].Load(uig,ujrc); LOOPs((*prSetSubNext[urSet][vrSet]),ie) {NextInsert(bpiNext,ie); nonLkct[ie]++; strayE[nstrayE].Load(ie,ujrc); strayE[nstrayE].bol = TRUE; nstrayE++;}} if(prSetSubNext[vrSet][urSet]->nele) {strayG[nstrayG++].Load(vig,vjrc); LOOPs((*prSetSubNext[vrSet][urSet]),ie) {NextInsert(bpiNext,ie); nonLkct[ie]++; strayE[nstrayE].Load(ie,vjrc); strayE[nstrayE].bol = FALSE; nstrayE++;}}}} else {strayG[nstrayG++].Load(uig,ujrc); LOOPs(rSetieNext[urSet],ie) {NextInsert(bpiNext,ie); nonLkct[ie]++; strayE[nstrayE].Load(ie,ujrc); strayE[nstrayE].bol = TRUE; nstrayE++;} strayG[nstrayG++].Load(vig,vjrc); LOOPs(rSetieNext[vrSet],ie) {NextInsert(bpiNext,ie); nonLkct[ie]++; strayE[nstrayE].Load(ie,vjrc); strayE[nstrayE].bol = FALSE; nstrayE++;}}} strayG[nstrayG++] = rcs[twnLink-1]; rcs[twnLink-1].Get(ig,jrc); nstrayEanchorStart = nstrayE; ieLOOP {NextInsert(bpiNext,ie); nonLkct[ie]++; strayE[nstrayE].Load(ie,jrc); strayE[nstrayE].bol = FALSE; nstrayE++;} multinonLk = FALSE; iLOOPs(bpiNext) if(1<nonLkct[i]) {multinonLk = TRUE; break;} iLOOPs(bpiNext) ZEROOUT(bpFacVar[i][0],nrc); for(i=0;i<nstrayE;i++) {strayE[i].Get(ie,jrc); bpFacVar[ie][jrc] = TRUE;} BPreadybase(); prec qtcur = snqtMax; TWPushThru(qtcur); TWevalMV(bpcur); if(bpcur.netMV < -(TOLERANCEV) && 0 < ZeroPress(qtcur)) {prec lastTipOverCheckqtlo = -1; qthi = qtcur; qtlo = 0; do {qtcur = (qtlo + qthi) * 0.5; TWPushThru(qtcur); TWevalMV(bpcur); if(bpcur.netMV < -(TOLERANCEV)) {qthi = qtcur; if(!ZeroPressQ(qthi)) {twcond = abortChoke; break;}} else if(TOLERANCEV < bpcur.netMV) {qtlo = qtcur;} else break;} while(!IsEqualQ(qthi,qtlo));} TWQuantSufficient(); snqtMax =
```

```
qtcur;} void TWDoBlockage() {int iga, igs, jrc; if(twcond ==
abortChoke) {rca[0].Get(iga, jrc); rcs[0].Get(igs, jrc);
twpm.BlockPair(igs, iga, jrc); if(!twChoke.j) {rca[0].Get(iga,
jrc); twpm.Blockigs(iga, jrc); twChoke.i--;} while(0<=twChoke.i)
{rcs[twChoke.i].Get(igs, jrc); twpm.Blockiga(igs,jrc);
twChoke.i--;}} if(twcond == abortPair) {rca[0].Get(iga, jrc);
rcs[0].Get(igs, jrc); twpm.BlockPair(igs, iga, jrc);}} void
TWTryIteration() {TWGensnqtMax(); if(twcond == normalTWCond)
{TWLoadBPlayEquateMvMc(); if(twcond == normalTWCond) {prec
cpvalue = CTGetProfit(); BPlay(TWbpFill, (TOLERANCE), snqtMax);
TWSwapQin(); BPFin(FALSE); if(cpvalue < CTGetProfit()) {} else
{TWSwapQin(); BPFin(FALSE); twcond = abortPair;}}}} void
TWdoFundamentalTW(BOOL& get2ndpair, BOOL& raCurrent)
{TWTryIteration(); if(twcond != normalTWCond && !raCurrent)
{RCRoundAdjustment(FALSE); raCurrent = TRUE; twcond =
normalTWCond; TWTryIteration();} if(twcond != normalTWCond)
{TWDoBlockage(); get2ndpair = TRUE;}} void
TWdoCleanCycleTW(BOOL& get2ndpair) {TWGensnqtMax(); if(twcond ==
normalTWCond) {if(twcase == splitcyVer) {} else
TWLoadBPlayEquateMvMc(); if(twcond == normalTWCond)
{TWSwapQin(); LoadCTbOrg(); CTMakeFeasible();}} if(twcond !=
normalTWCond) {TWDoBlockage(); get2ndpair = TRUE;}}
#define hiCheck if(hi.x < TOLERANCE) {twcond = abortChoke;
return;}
void TWMaxsplitcyYield(void (FnEval)(Pairprec& coordinate))
{Pairprec org,
cur; Pairprec lo (0,0); Pairprec md ; Pairprec hi (snqtMax , -
1.0);
Pairprec sd (snqtMax - twToleranceClamp , -1.0); FnEval(hi);
FnEval(sd);
if(hi.y < sd.y || !hi.y) {do {md.x = hi.x * 0.5f; FnEval(md);
if(!md.y)
{hi = md; hiCheck;}} while(!md.y); org = hi;
while(!IsEqual(lo.x, hi.x,
twToleranceClamp)) {if(md.x - lo.x < hi.x - md.x) {cur.x = (md.x
+ hi.x)
* 0.5; FnEval(cur); if(md.y < cur.y) {lo = md; md = cur;} else
{hi = cur;
hiCheck;}} else {cur.x = (lo.x + md.x) * 0.5; FnEval(cur);
if(md.y < cur.y) {hi = md; hiCheck; md = cur;} else {lo =
cur;}}}
if(lo.y < org.y) cur = org; else cur = lo;
if(cur.y < TOLERANCE || cur.x < TOLERANCE) twcond = abortChoke;
else snqtMax = cur.x;}}
#undef hiCheck
void TWsplitcyVerYield(Pairprec& coordinate) {int i; int uig,
ujrc;
int vig, vjrc; prec snqt = coordinate.x; prec quant =
coordinate.x;
for(i=twnLink-1; iSplitVer<i; i--) {rca[i].Get(uig,ujrc); rcs[i-
1]
```

```
.Get(vig,vjrc);    quant    =    TWvfu(RCU,RCU.allotment,    RCV,
RCV.allotment, quant)
;} quant -= snqt; if(0 < quant) coordinate.y = quant;
else coordinate.y = 0; return;}
void TWsplitcyHorYield(Pairprec& coordinate) {int i; int uig,
ujrc;
int vig, vjrc; prec snqt = coordinate.x; prec quant =
coordinate.x;
for(i=twnLink-1; iSplitHor<i; i--) {rca[i].Get(uig,ujrc); rcs[i-
1]
.Get(vig,vjrc);    quant    =    TWvfu(RCU,RCU.allotment,    RCV,
RCV.allotment, quant)
;} quant -= TWufv(RCsph, RCsph.allotment, vEnd, vEnd.allotment,
snqt);
if(0 < quant) coordinate.y = quant; else coordinate.y = 0;
return;}
void TWGensnqtMax() {int iLink; int ig, jrc; int uig, ujrc; int
vig, vjrc;
prec        snqtMaxhold;        snqtMax        =        prec_MAX;
for(iLink=0;iLink<twnLink;iLink++)
{prec qt; rca[iLink].Get(ig, jrc); qt = GIncMax(ig, jrc);
if(qt  <=  snqtMax)  {snqtMax  =  qt;  twChoke.Load(iLink,0);}
rcs[iLink]
.Get(ig, jrc); qt = GDecMax(ig, jrc); if(qt <= snqtMax) {snqtMax
= qt;
twChoke.Load(iLink,1);}    if(iLink    !=    twnLink    -    1)
{rcs[iLink].Get(vig,
vjrc);  rca[iLink+1].Get(uig,  ujrc);  snqtMax  =  TWufv(RCU,
RCU.allotment,
RCV, RCV.allotment, snqtMax);} if(snqtMax < TOLERANCE)
{twcond = abortChoke; return;}} if(twcase == splitcyVer)
{snqtMaxhold = snqtMax; TWMaxsplitcyYield(TWsplitcyVerYield);
if(snqtMax < snqtMaxhold) twChoke.Load(iSplitVer,1);}
else if(twcase == splitcyHor) {snqtMaxhold = snqtMax;
TWMaxsplitcyYield(TWsplitcyHorYield); if(snqtMax < snqtMaxhold)
twChoke.Load(iSplitHor,0);} if(twcond == normalTWCond)
{TWPushThru(snqtMax); TWQuantSufficient();}}
void TopWalkExit(BOOL& profitable) {int rtCond;
profitable    =    twpm.WalkProfitable();    GenNextMV(jrcNext);
if(profitable)
AxisWalk(rtCond);} void TopWalk(int& rtCond, BOOL& profitable,
int lwControl) {static long rcRActLoc = -2; long twCtDw = iLtw;
rtCond = 0; BOOL raCurrent = FALSE; TWGenitopNext(lwControl);
BOOL twContinue = (itopNext.nele || lwControl)? TRUE : FALSE;
if(!lwControl) twpm.WalkInit(); while(twContinue) {int ig, jrc;
int iga,
igs;    BOOL    get2ndpair    =    FALSE;    if(!lwControl)
twpm.Clear(jrcNext);
else if(lwControl ==2) {get2ndpair = TRUE; lwControl = 1;}
do {twcond = normalTWCond; if(get2ndpair) {twpm.GetBestPair(igs,
iga, jrc)
; if(igs == -1) get2ndpair = FALSE;}
```

```
if(!get2ndpair || rcRActLoc != rcRoundAdjustmentCt)
{rcRActLoc = rcRoundAdjustmentCt; TWGenitopNext(lwControl);
if(!itopNext.nele  &&  !lwControl)  {TopWalkExit(profitable);
return;}
GenNextMV(jrctwNext); TWGenMVs(); twpm.ItPrep();
jrctwLOOP              {twpm.ItPrepCol(jrc);                 igwLOOP
{twpm.ItNoteAdd(RCG.gmva);
twpm.ItNoteSub(RCG.twgmvs);}} twpm.GetBestPair(igs, iga, jrc);}
get2ndpair = FALSE; twCtDw--; if(igs != -1) {if(RCS.twuig == -1)
{int rtCond2; if(!lwControl) {GtoGTransfer(igs, iga, jrc);} else
{rca[0]
.Load(iga,jrc); rcs[0].Load(igs,jrc); twnLink = 1; iSplitVer = -
1;
iSplitHor   =   -1;   twcase   =   gtogNullLink;   RCS.twQuant   =
RCS.allotment;
RCA.twQuant = RCA.allotment; GtoGTransfer(igs, iga, jrc);
twpm.SetTemp(igs, iga, jrc); return;} GenNextMV(jrcNext);
AxisWalk(rtCond2); raCurrent = FALSE;} else {TWLoadrcarcs(iga,
igs, jrc);
if(twcond == normalTWCond) {TWCheckCleanCycle(); twpm.PrePair();
if(!cleanCycle) TWdoFundamentalTW(get2ndpair, raCurrent);
else TWdoCleanCycleTW(get2ndpair); twpm.PostPair(igs, iga, jrc);
if(!get2ndpair)    {raCurrent    =    FALSE;    if(lwControl)
{twpm.SetTemp(igs, iga,
jrc); return;}}} else {TWDoBlockage(); get2ndpair = TRUE;}}}}
while(igs != -1 && 0<=twCtDw); if(!lwControl)
twContinue = (twpm.ProfitableSinceBlocked() && 0<=twCtDw);
else      {twpm.SetTemp(-1,      -1,      -1);        return;}}
TopWalkExit(profitable);}
BOOL   TW0SplitGroup()   {int   i,   jrc;   int   itop;   int
baseRef[RCDTMAX_NRC];
TWGenitopNext(0); LOOPs(itopNext, itop) {SPREAD(baseRef,nrc,-1);
LOOPs(horNextw[itop],jrc) baseRef[jrc] = rc[itop][jrc].igUp; i =
itop+1;
RESUMELOOPs(itopNext,i)  LOOPs(horNextw[i],jrc)  if(baseRef[jrc]
== rc[i]
[jrc].igUp) return TRUE;} return FALSE;} void RWPrep() {int i;
NextPrep(sbiNext,     mrc);      NextPrep(adiNext,      mrc);
NextPrep(jrcrwNextBase,
nrc); for(i=0;i<mrc;i++) {NextPrep(dpTieNext [i], nrc);
NextPrep(dpTieNextDol[i], nrc);} NextPrep(jrcbusNext, nrc);}
void  RWPrepWalkSession()  {rwiHAT  =  -1;}  void  RWbpFill(BPds
&bbs,prec f)
{int       i,       ig,       ie;       int       jrc;
ARRAYCOPY(bpbase.bOrg[0],bbs.bOrg[0],mrc);
dolBus = 0; jrcbusLOOP {bus[jrc] = f/oldMC[jrc] - oldQuant[jrc];
if(bus[jrc]  <  0)  bus[jrc]  =  0;  if(!IsDolCol(jrc))  {ieLOOPs
{bbs.bOrg[ie]
*=  RCs.factor  -  RCs.dedrs  *  bus[jrc];  if(bbs.bOrg[ie]  <  0)
{bbs.bOrg[ie]
= 0;}}} else dolBus += bus[jrc]; ieLOOPa bbs.bOrg[ie]
```

```
*=  RCa.factor  +  RCa.dedra  *  bus[jrc];}  if(rwDolAct)
{dolHCw.GetSub(ig,
jrc); ieLOOP bbs.bOrg[ie] *= RCG.factor - RCG.dedrs * dolBus;}
iLOOPs(bpiNext) bbs.bOrg[i] *= potentialCTwt[i]; bbs.e = f;}
void SetsubBlk(int ie, int jrc, BOOL cond, BOOL allElements
/*=TRUE*/)
{int ig = rc[ie][jrc].igUp; if(!allElements) rc[ie][jrc]
.subBlk += cond ? 1 : -1; else ieLOOP rc[ie][jrc].subBlk += cond
? 1 : -1;}
void SwapsubBlk(HCol& hcol) {int ig, ie, jrc; int k; ResConduit*
prc;
prcLOOPhcigjrc(hcol)      ieLOOP      {Swap(rc[ie][jrc].subBlk,
rc[ie][jrc]
.subBlkHold);}} void SetRWiHATBlocking(BOOL cond) {int jrc;
LOOPs(horNextw[rwiHAT],jrc)   SetsubBlk(rwiHAT,   jrc,   cond,
FALSE);}
#define sliceMaxs (rc[IGs][jrc].dirPut ? prec_MAX : rwSliceMax /
rc[IGs]\
[jrc].dedrs)
#define sliceMaxa (rc[IGa][jrc].dirPut ? prec_MAX : rwSliceMax /
rc[IGa]\
[jrc].dedra)
#define sliceMina (rc[IGa][jrc].dirPut ? rwSliceMin * qOrg[jrc]\
:(rwSliceMin)/(rc[IGa][jrc].dedra))
void RWDPDrag(int ipull, BOOL firstPass/*=FALSE*/)
{static NEXTs jrcNextDPsubBlk; static HColIn dolDPsubBlk;
static int ctPullisSourceForDP[RCDTMAX_NRC]; if(firstPass)
{NextPrep(jrcNextDPsubBlk,   nrc);   dolHCw.InInit(dolDPsubBlk,
FALSE);
ZEROOUT(ctPullisSourceForDP[0], nrc);} int igs, jrc; int ggCt =
-1;
BOOL shortDP = FALSE; BOOL shortPull = FALSE; DolBalance();
gtogTransferOKSwapDirection = FALSE; while(TRUE) {ggCt++;
prec qtPull = CTGetbOrg(ipull); prec qtDP = prec_MAX; int qtDPig
= -1;
int qtDPjrc = -1; LOOPs(dpTieNext[ipull], jrc)
{prec qt = CTGetBound(dpTieb[ipull][jrc], ipull); if(qt < qtDP)
{qtDP = qt; qtDPig = dpTieb[ipull][jrc]; qtDPjrc = jrc;}}
if(dpTieNextDol[ipull].nele)  {prec  qt  =  CTGetBound(dolRow,
ipull);
if(qt < qtDP) {qtDP = qt; qtDPig = dolRow; qtDPjrc = dolCol;}}
if(!shortDP  &&  !shortPull)  {if(qtDP  +  TOLERANCE  <  qtPull)
shortDP = TRUE;
else if(qtPull + TOLERANCE < qtDP) shortPull = TRUE;}
if(shortDP  &&  qtDP  +  TOLERANCE  <  qtPull)  {SetsubBlk(qtDPig,
qtDPjrc, TRUE)
;   jrc   =   qtDPjrc;   NextInsert(jrcNextDPsubBlk,   jrc);
if(!IsDolCol(jrc))
{headHCw[jrc].SubFind(TRUE);        if(headHCw[jrc].Goods())
{headHCw[jrc]
.GetSub(igs,  jrc);  GtoGTransfer(igs,  jrc,  qtDPig,  qtDPjrc,
FALSE, TRUE);
```

```
if(igs == IGa) if(++ctPullisSourceForDP[jrc] == 64)
SetsubBlk(igs, jrc,
TRUE);} else break;} else {dolHCw.In(dolDPsubBlk, qtDPig,
qtDPjrc);
dolHCw.SubFind(TRUE, qtDPig, qtDPjrc); dolHCw.SetAdd(qtDPig,
qtDPjrc);
if(dolHCw.Goodsa()) {DolGoGTransfer(TRUE); dolHCw.GetSub(igs,
jrc);
if(igs == IGa) if(++ctPullisSourceForDP[jrc] == 64)
SetsubBlk(igs, jrc,
TRUE);} else break;}} else if(shortPull && qtPull + TOLERANCE <
qtDP)
{BOOL didTrans = FALSE; LOOPs(jrcNextDPsubBlk, jrc)
if(!IsDolCol(jrc))
{int iga = IGa; int igs = dpTieb[ipull][jrc];
Swap(rc[igs][jrc].subBlk,
rc[igs][jrc].subBlkHold); GenGroupMV(igs, jrc); GenGroupMV(iga,
jrc);
if(rc[igs][jrc].gmvs < rc[iga][jrc].gmva) {GtoGTransfer(igs,
jrc, iga,
jrc, FALSE, TRUE); didTrans = TRUE;} Swap(rc[igs][jrc].subBlk,
rc[igs]
[jrc].subBlkHold);} else {BOOL profitable; long ctDown = 25;
SwapsubBlk(dolHCw); DolWalk(profitable, ctDown, &dolDPsubBlk,
&dolRWPull[ipull]); SwapsubBlk(dolHCw); didTrans = (ctDown !=
25);}
if(!didTrans) break;} else {break;}} gtogTransferOKSwapDirection
= TRUE;
DolBalance(); if((rwsliceTrigger < rowFactor[ipull] ||
(firstPass && ggCt)
) && !iLTimer.Elapse(iLrwTime)) {int rtCond2;
rwsliceTrigger = rowFactor[ipull] + rwSliceFac;
SetRWiHATBlocking(TRUE);
GenNextMV(jrcNext); ATLManager(rtCond2);
SetRWiHATBlocking(FALSE);
DolBalance();} GenNextMV(jrcrwNextBase); if(rwbestProfit <
CTGetProfit())
{rwProfitable = TRUE; rwbestProfit = CTGetProfit();
rwbestImage.Out();}}
void RidgeWalk(int& rtCond, BOOL& profitable, int ipull) {int i,
ie, jrc;
prec minf, maxf; if(ipull==dolRow) {DolRidgeWalk(profitable);
rwProfitable = profitable; rtCond = 0; return;} prec rprofit=0;
rwProfitable = FALSE; NextClear(jrcrwNextBase);
NextClear(adiNext);
LOOPs(dpTieNext[ipull],jrc) {IGa = -1;} rwDolAct = FALSE;
LOOPs(horNextw[ipull],jrc) {IGa = rc[ipull][jrc].igUp;
NextInsert(jrcrwNextBase,jrc); ieLOOPa NextInsert(adiNext,ie);
if(IsDolCol(jrc)) rwDolAct = TRUE;} if(!jrcrwNextBase.nele)
{rtCond = 0;
profitable = FALSE; return;} rwbestProfit = CTGetProfit();
rwbestImage.Out(); rwpm.WalkInit(); rwiHAT = ipull;
```

```
rwsliceTrigger = rowFactor[ipull] + rwSliceFac; RWDPDrag(ipull,
TRUE);
while(rowFactor[ipull] < maxRowFac[ipull]) {BOOL didGtoG;
NextCopy(jrcrwNextBase,jrcbusNext);     didGtoG     =     FALSE;
if(rwDolAct)
{if(NextOverlap(jrcbusNext, jrcNextDol)) {dolHCw.SubFind(TRUE,
&dolRWPullRev[ipull]); if(!dolHCw.Goods()) rwDolAct = FALSE;}
else     rwDolAct     =     FALSE;}     LOOPs_del(jrcbusNext,jrc)
{if(!IsDolCol(jrc))
{headHCw[jrc].SubFind(FALSE, IGa, jrc); if(headHCw[jrc].Goods())
headHCw[jrc].GetSub(IGs, jrc); if(!headHCw[jrc].Goods()
|| RCa.ir == RCa.nir) {NextDelete(jrcbusNext,jrc);
NextDelete(jrcrwNextBase,jrc);}
else    if(RCa.gmva    <    RCs.gmvs    &&    ZeroPressV(RCs.gmvs))
{oldQuant[jrc]
= RCa.factor/RCa.dedra; oldMC [jrc] = RCs.gmvs;}
else {prec maxquant = sliceMaxa; prec minquant = sliceMina;
GtoGTransfer(IGs,IGa,jrc,FALSE,TRUE,minquant,maxquant);
RWDPDrag(ipull);
GenNextMV(jrcrwNextBase);   didGtoG   =   TRUE;   break;}}   else
{if(rwDolAct)
{if(RCa.ir       ==       RCa.nir       ||       !dolHCw.Goods())
{NextDelete(jrcbusNext,jrc);
NextDelete(jrcrwNextBase,jrc);}
else   if(RCa.gmva   <   RCDs.gmvs   &&   ZeroPressV(RCDs.gmvs))
{oldQuant[jrc]
=  RCa.factor/RCa.dedra;  oldMC  [jrc]  =  RCDs.gmvs;}  else
{dolHCw.SetAdd(IGa,
jrc);           DolGoGTransfer(TRUE);            RWDPDrag(ipull);
GenNextMV(jrcrwNextBase);
didGtoG = TRUE; break;}} else {NextDelete(jrcbusNext,jrc);}}}
if(!didGtoG)
{if(!jrcbusNext.nele) break; if(1 < jrcbusNext.nele) {minf =
0.0f;
maxf = prec_MAX; prec dolMax = 0; int ig; if(rwDolAct) {int
rwDolCt = 0;
jrcbusLOOP if(IsDolCol(jrc)) rwDolCt++; dolHCw.GetSub(ig, jrc);
dolMax = GDecMax(ig, jrc)/rwDolCt;} jrcbusLOOP {prec mine =
(oldQuant[jrc]
+ oldQuant[jrc] + sliceMina) * oldMC[jrc]; minf = __max(minf,
mine);
prec maxe[5]; maxe[0] = !IsDolCol(jrc) ? GDecMax(IGs,jrc) :
dolMax;
maxe[1] = GIncMax(IGa,jrc); maxe[2] = !IsDolCol(jrc) ? sliceMaxs
: dolMax;
maxe[3]  =  sliceMaxa;  for(int  ii=1;ii<4;ii++)  if(maxe[ii]  <
maxe[0])
maxe[0]  =  maxe[ii];  maxe[4]  =  (maxe[0]  +  oldQuant[jrc])  *
oldMC[jrc];
maxf = __min(maxf, maxe[4]);} NextClear(sbiNext);
jrcbusLOOP  if(!IsDolCol(jrc))  ieLOOPs  NextInsert(sbiNext,ie);
if(rwDolAct)
```

```
{dolHCw.GetSub(ig,    jrc);    ieLOOP   NextInsert(sbiNext,ie);}
NextCopy(sbiNext,
bpiNext);    NextAdd(bpiNext,    adiNext);    iLOOPs(bpiNext)
ZEROOUT(bpFacVar[i]
[0],nrc);    jrcbusLOOP    {if(!IsDolCol(jrc))    ieLOOPs
bpFacVar[ie][jrc] = TRUE;
ieLOOPa    bpFacVar[ie][jrc]    =    TRUE;}    if(rwDolAct)
{dolHCw.GetSub(ig, jrc);
ieLOOP bpFacVar[ie][jrc] = TRUE;} BPreadybase(); BPlay(RWbpFill,
minf,
maxf);  jrcbusLOOP  {if(!IsDolCol(jrc))  IncAllotment(IGs,jrc,-
bus[jrc],
FALSE); IncAllotment(IGa,jrc, bus[jrc], FALSE);} if(rwDolAct)
{dolHCw.GetSub(ig, jrc); IncAllotment(ig, jrc, -dolBus, FALSE);}
BPFin();
RWDPDrag(ipull);} else {jrc = jrcbusNext.lo; if(!IsDolCol(jrc))
{prec  maxquant  =  sliceMaxa;  prec  minquant  =  sliceMina;
GtoGTransfer(IGs,
IGa,jrc,FALSE,TRUE,minquant,maxquant);} else {dolHCw.SetAdd(IGa,
jrc);
DolGoGTransfer(TRUE);}          RWDPDrag(ipull);}}          else
{RWDPDrag(ipull);}}
rwbestImage.In(); rwiHAT = -1; int ig; LOOPs(jrcNext, jrc)
igwLOOP   ieLOOP   RCE.subBlk   =   RCE.subBlkHold   =   0;
GenNextMV(jrcNext);
if(rwProfitable) {BOOL rtCond2; ATLManager(rtCond2);
profitable = rwpm.WalkProfitable();} else profitable = FALSE;
rtCond = 0;}
void LWPrep() {NextPrep(jrclwNext,nrc);}
void LWSetFactor(prec setlwFactor) {lwFactor = setlwFactor;}
prec LWGetfacIn() {return lwFactor;} void LWInitLat() {int i;
lwOrgCTimage.Out(2);     lwOrgProfit     =     CTGetProfit();
iLOOPs(ilwNext)
{prec facOut; CTShiftBOrgIn(i, lwFactor, facOut); lwLatpotential
[i]
=   potential   [i]   *   facOut;   lwLatpotentialCTwt[i]   =
potentialCTwt[i]
* facOut;} lwLatCTimage.Out(2); LWRestoreLatCtMar(FALSE);}
void   LWRestoreLatCtMar(BOOL  ctImageAlso/*=TRUE*/)  {int   i;
if(ctImageAlso)
lwLatCTimage.In();    iLOOPs(ilwNext)    {potential    [i]    =
lwLatpotential [i];
potentialCTwt[i]  =  lwLatpotentialCTwt[i];  ffp[i].potential  =
potential[i];
GenRowFactor(i);}}    void    LWRestoreOrgCtMar()    {int    i;
lwOrgCTimage.In();
iLOOPs(ilwNext)   {potential   [i]   =   lwOrgpotential   [i];
potentialCTwt[i]
=  lwOrgpotentialCTwt[i];  ffp[i].potential  =  potential[i];
GenRowFactor(i);
}} void LateralWalk(BOOL& profitable) {long lwCtDw = iLtw; int
i;
```

```
lwpm.WalkInit(); LWInitLat(); do {int i, iga, igs, jrc; int
lwControl = 1;
lwpm.Clear(jrcNext); do {int rtCond; BOOL profitable2;
TopWalk(rtCond,
profitable2, lwControl); lwControl = 1;
lwpm.GetTemp(igs,iga,jrc);
LWRestoreOrgCtMar(); lwCtDw--; if(igs != -1) {LoadCTbOrg();
CTMakeFeasible(); if(lwOrgProfit < CTGetProfit()) {if(lwCtDw)
LWInitLat();
} else {lwpm.BlockPair(igs,iga,jrc); lwControl = 2; TWSwapQin();
iLOOPs(irfNextw) if(!twGenFactorNext.use[i]) GenRowFactor(i);
if(lwCtDw)
LWRestoreLatCtMar(); else LWRestoreOrgCtMar();}}}
while(igs != -1 && lwCtDw);} while(lwpm.ProfitableSinceBlocked()
&& lwCtDw); for(i=0;i<mrc;i++) {} GenNextMV(jrcNext);
profitable = lwpm.WalkProfitable(); return;} long atlLimit;
void ATLManager(int& rtCond) {BOOL reDoAxis = FALSE; BOOL
profitable;
AxisWalk(rtCond); TopWalk(rtCond, profitable, FALSE);
reDoAxis |= profitable; LateralWalk(profitable); reDoAxis |=
profitable;
while(profitable && 0 < atlLimit--) {if(profitable)
TopWalk(rtCond,
profitable, FALSE); reDoAxis |= profitable; if(profitable)
LateralWalk(profitable); reDoAxis |= profitable;}
AxisWalk(rtCond);}
void WalkReadyInteration() {atlLimit = 100;
dolHCw.BlockClear();}
void Walk() {BOOL profitable; int i, lmi; int j; int rtCond;
BOOL startCycle = TRUE; iLTimer.Init();
long rwCtDw = iLrw * ipullNext.nele; long dolBlkCt = 0;
iLpm.PermBlockInit(); iLpm.WalkInit(); iLpm.Clear(jrcNext);
for(j=0;j<nrc;
j++) iLggColumnTolerance[j] = qOrg[j] * iLggFactorTolerance;
iLawCt = 0;
iLtwCt = 0; iLlwCt = 0; awpm.PermBlockInit();
twpm.PermBlockInit();
permanentBlockPair = FALSE; permBlockPairCt = 0;
TWPrepWalkSession();
RWPrepWalkSession(); WalkReadyInteration(); ATLManager(rtCond);
dolBlkCt += dolHCw.prematureBlockCT; ExplodeWalk(rtCond);
if(ipullNext.nele && rwCtDw && !iLTimer.Elapse(iLrwTime))
{lmi = i = ipullNext.lo; do {if(i == ipullNext.lo)
{if(startCycle)
{wkpm.WalkInit(); startCycle = FALSE;} else
if(!wkpm.WalkProfitable())
break;} if(rowFactor[i] < maxRowFac[i]) {WalkReadyInteration();
RidgeWalk(rtCond, profitable, i); dolBlkCt +=
dolHCw.prematureBlockCT;
if(profitable) lmi = i;} NextAround(ipullNext, i);}
while(lmi != i && --rwCtDw && !iLTimer.Elapse(iLrwTime));}}
```

```
void LoadHC(NEXTs& jNext, HCol& hcol) {int ig, jrc;
hcol.NoteInit();
LOOPs(jNext, jrc) igbLOOP hcol.Note(&rc[ig][jrc], ig, jrc);
hcol.NoteFin()
;} void MaxFreshPrep(BOOL weight) {int ig, jrc; int i,j; NEXTs
wnext;
initialLoad = FALSE; LoadHC(jrcNextDol, dolHCb); if(dolAct)
{ResConduit* prc; int k, ig, jrc, ir; prcLOOPhcigjrc(dolHCb)
{RCG.allotment *= dolPrice[jrc]; if(ig != dolRow) for(ir=0; ir
<= RCG.nir;
ir++) {RCG.rstop[ir] *= dolPrice[jrc]; RCG.dedr [ir] /=
dolPrice[jrc];}}
ig = dolRow; LOOPs(jrcNextDol, jrc) RCG.gmva = 1;
CTNoteDolProfitAlso();}
NextPrep(jrcNextHCb,nrc); for(jrc=0;jrc<nrc;jrc++)
{if(!IsDolCol(jrc))
{NEXTs tempNext; NextPrep1(tempNext, jrc); LoadHC(tempNext,
headHCb[jrc]);
NextInsert(jrcNextHCb, jrc);} else {if(jrc != dolCol)
{headHCb[jrc]
.NoteInit();} else {headHCb[jrc].Load(dolHCb);
NextInsert(jrcNextHCb, jrc);
}}} if(TRUE) {int idp, jProd; SPREAD2d(dpTieb, mrc, nrc, -1);
for(jProd=0;
jProd<rcmProd; jProd++) LOOPs(dpNext, idp)
if(ZeroPress(CTGetOrgaElement(idp, jProd))) {jrc =
horNextb[idp].lo;
dpTieb[jProd][jrc] = idp;}} SPREAD(qOrgwtIn, nrc, 1.0f);
SPREAD(qOrgwtOt,
nrc, 1.0f); if(weight) {for(jrc=0;jrc<nrc;jrc++)
if(headNextb[jrc]
.nele && qOrg[jrc] && !IsDolCol(jrc))
{prec tarQuant = 50.0f * headNextb[jrc].nele; qOrgwtIn[jrc]
= tarQuant/qOrg[jrc]; qOrgwtOt[jrc] = qOrg[jrc]/tarQuant;
igbLOOP {RCG.allotment *= qOrgwtIn[jrc]; for(int ir=0;
ir<=RCG.nir; ir++)
{RCG.rstop[ir] *= qOrgwtIn[jrc]; RCG.dedr [ir] /=
qOrgwtIn[jrc];}}}}
WeightqOrg(); for(i=0;i<nrSet;i++) for(j=0;j<nrSet;j++)
{NextIntersect(wnext,rSetieNext[i],rSetieNext[j]);
if(wnext.nele)
{prSetInterNext[i][j] = new NEXTs; NextCopy(wnext,
*prSetInterNext[i][j]);
NextCopy(rSetieNext[i], wnext); NextSub(wnext, rSetieNext[j]);
prSetSubNext[i][j] = new NEXTs; NextCopy(wnext,
*prSetSubNext[i][j]);}
NextUnion(wnext,rSetieNext[i],rSetieNext[j]); if(TRUE)
{prSetUnionNext[i]
[j] = new NEXTs; NextCopy(wnext, *prSetUnionNext[i][j]);}}
for(i=0;i<mrc;
i++) NextPrep(gOverLapRowNext[i],mrc); for(int
irSet=0;irSet<nrSet;
```

```
irSet++) {int ie1, ie2; LOOPs(rSetieNext[irSet], ie1)
LOOPs(rSetieNext[irSet], ie2) if(ie1 != ie2)
{NextInsert(gOverLapRowNext[ie1],                       ie2);
NextInsert(gOverLapRowNext[ie2],
ie1);}} BPPrep(); AWPrep(); TWPrep(); RWPrep(); LWPrep();
SetRCType();
LOOPs(jrcNextHCb,    jrc)    {HCol    tHC;    RCFilter    rcf;
rcf.Include(rctFix);
prec qtFix = headHCb[jrc].GetSumAllot(rcf); rcf.SetButFix();
tHC.Load(headHCb[jrc],rcf);  tHC.Apportion(qOrg[jrc]  -  qtFix,
FALSE);}
SetMarginCtlW(TRUE, FALSE); GenRowFactor(); LoadCTbOrg();
CTFinLoad(weight); SPREAD(roundAdjustmentOK, nrc, TRUE);
SetMarginCtlW(TRUE,      FALSE);}      void      MaxFresh(BOOL
weight/*=FALSE*/)
{MaxFreshPrep(weight);     CTMaximize();     GenNextMV(jrcNext);
Walk();}
void MaxPotential(prec *potentialNew, BOOL doWalk /*=TRUE*/)
{ARRAYCOPY(potentialNew[0],       potential[0],          mrc);
SetMarginCtlW(TRUE, FALSE)
; LoadCTbOrg(); MakeCTFeasible(); GenNextMV(jrcNext); if(doWalk)
Walk();}
void     WeightqOrg()      {for(int       jrc=0;jrc<nrc;jrc++)
if(!IsDolCol(jrc))
qOrg[jrc] *= qOrgwtIn[jrc]; else if(jrc != dolCol) qOrg[jrc] = -
1.0;}
void MaxqOrg(prec *qOrgNew) {int jrc; SetMarginCtlB();
ARRAYCOPY(qOrgNew[0],qOrg[0],nrc);  WeightqOrg();  SetRCType();
XctMVPre();
LOOPs(jrcNextHCb, jrc) {prec residual = qOrg[jrc] - headHCb[jrc]
.GetSumAllot(); prec bus; if(ZeroPressQ(residual)) {HCol tempHC;
RCFilter rcf; rcf.SetButFix(); tempHC.Load(headHCb[jrc], rcf);
if(residual<0)                   while(ZeroPressQ(residual)<0)
{tempHC.SubFind(TRUE);
if(tempHC.Goods()) {int ig, jrc; tempHC.GetSub(ig,jrc);
bus  =  __min(-residual,GDecMax(ig,jrc));  residual  +=  bus;
IncAllotment(ig,
jrc,-bus, TRUE); LoadCTbOrg(); MakeCTFeasible();} else {residual
= 0;}}
else while(0<ZeroPressQ(residual)) {tempHC.GenColMV(); int k;
ResConduit* prc; prcLOOPhc(tempHC) if(prc->ir == prc->nir)
prc->gmva  =  -  1.0;  tempHC.AddFind(FALSE);  if(tempHC.Gooda())
{int ig, jrc;
tempHC.GetAdd(ig,jrc); bus = __min(residual,GIncMax(ig,jrc));
residual -= bus; IncAllotment(ig,jrc, bus, TRUE); LoadCTbOrg();
MakeCTFeasible();} else {residual = 0;}}}} XctMVPost();
SetMarginCtlW(TRUE,   FALSE);   GenRowFactor();   LoadCTbOrg();
MakeCTFeasible()
;  GenNextMV(jrcNext);  Walk();}  void  MaxPotentialqOrg(prec
*potentialNew,
prec     *qOrgNew)     {MaxPotential(potentialNew,     FALSE);
MaxqOrg(qOrgNew);}
```

```
BOOL GetPermanentBlockPair() {return permanentBlockPair;}
void ConvDolToReal(int jrc, prec& gmvs, prec& gmva, prec&
qtAllot)
{if(IsDolCol(jrc) && jrc != dolCol) {gmvs *= dolPrice[jrc];
gmva *= dolPrice[jrc]; if(ZeroPress(dolPrice[jrc]))
qtAllot /= dolPrice[jrc];}} ResConduit* GetpGroup(int ig, int
jrc)
{return &rc[ig][jrc];} void GetGroupMV(int ig, int jrc, prec&
gmvs,
prec& gmva, prec& qtAllot) {gmvs = RCG.gmvs / qOrgwtOt[jrc];
gmva = RCG.gmva / qOrgwtOt[jrc]; qtAllot = RCG.allotment *
qOrgwtOt[jrc];
ConvDolToReal(jrc, gmvs, gmva, qtAllot);} void GetGrouptwMV(int
ig,
int jrc, prec& twgmvs, prec& twgmva, prec& qtAllot)
{twgmvs = RCG.twgmvs / qOrgwtOt[jrc]; twgmva = RCG.twgmva /
qOrgwtOt[jrc];
qtAllot = RCG.allotment * qOrgwtOt[jrc]; ConvDolToReal(jrc,
twgmvs,
twgmva, qtAllot);} void RCGetRowMC(int ie, BOOL infMC, int&
rtCond,
prec& mc) {int jrc; int ig; prec mcs [RCDTMAX_NRC]; NEXTs hNext;
NextPrep(hNext,nrc); rtCond = 1; if(!ZeroPress(potential[ie]))
return;
LOOPs(horNextb[ie],jrc) {ig = RCE.igUp;
if((RCG.type == rctFix && !ZeroPress(RCG.factor)) ||
(!ZeroPress(qOrg[jrc]
* qOrgwtOt[jrc]))) return; mcs[jrc] = (!infMC) ? rcMVs[jrc] :
rctwMVs[jrc]
; if(bigM/1e7 < mcs[jrc]) return; if(ZeroPress(mcs[jrc]))
NextInsert(hNext,
jrc);} if(!hNext.nele) return; prec e1; prec ccPot1 = 1.0; int
eCt1 = 0;
LOOPs(hNext,jrc) {ig = RCE.igUp; if(RCG.type != rctFix)
{ccPot1 *= RCG.dedr[0]; ccPot1 *= 1.0/mcs[jrc]; eCt1++;}
else {ccPot1 *= RCG.factor;}} ccPot1 *= potential[ie];
e1 = pow(1.0/ccPot1, 1.0/eCt1); prec e2; prec ccPot2 = 1.0; int
eCt2 = 0;
LOOPs(hNext,jrc) {ig = RCE.igUp;
if(RCG.type != rctFix && !(e1 * (1.0/mcs[jrc]) <=
RCG.allotment))
{ccPot2 *= RCG.dedr[0]; ccPot2 *= 1.0/mcs[jrc]; eCt2++;}
else {ccPot2 *= RCG.factor;}} ccPot2 *= potential[ie];
e2 = pow(1.0/ccPot2, 1.0/eCt2); mc = 0.0; LOOPs(hNext,jrc) {ig =
RCE.igUp;
if(RCG.type != rctFix && !(e1 * (1.0/mcs[jrc]) <=
RCG.allotment))
{prec colQ = ((1.0/mcs[jrc]) * e2 - RCG.allotment); prec colC =
mcs[jrc];
colQ *= qOrgwtOt[jrc]; mc += colC * colQ;}} rtCond = 0;}
BOOL RCGetResultVec() {int i, ig, jrc; BOOL dif=FALSE;
SetMarginCtlB();
```

```
GenNextMV(jrcNext);        ZEROOUT(rcQuant        [0],        nrc);
ZEROOUT(rcQuantFix[0], nrc)
;  SPREAD(rcMVs,  nrc,  gmv_MAX);  SPREAD(rcMVa,  nrc,  0);
ZEROOUT(rctwMVa[0],
nrc); jrcLOOP igbLOOP if(ig != dolRow || jrc == dolCol) {rcQuant
[jrc]
+= RCG.allotment; if(RCG.type != rctFix) {if(rcMVs[jrc]
>  RCG.gmvs  &&  !rc[ig][jrc].IsMCsInfinite())  rcMVs[jrc]  =
RCG.gmvs;
if(rcMVa[jrc] < RCG.gmva) rcMVa[jrc] = RCG.gmva;}} if(dolAct)
{int ig,
jrc;  int  k;  ResConduit*  prc;  rcQuant[dolCol]  =  0;
prcLOOPhcigjrc(dolHCb)
if(ig  !=  dolRow  ||  jrc  ==  dolCol)  rcQuant[dolCol]  +=  prc-
>allotment;
dolHCb.SubFind(FALSE,       &dolFixRev);       if(dolHCb.Goods())
{dolHCb.GetSub(ig,
jrc);   rcMVs[dolCol]   =   RCG.gmvs;}   dolHCb.AddFind(FALSE,
&dolFixRev);
if(dolHCb.Gooda())  {dolHCb.GetAdd(ig,jrc);  rcMVa[dolCol]  =
RCG.gmva;}}
ARRAYCOPY(rcMVs[0], rctwMVs[0], nrc);
jrcLOOP  igbLOOP  RCG.twgmvs  =  RCG.gmvs;  TWGenitopNext(0);
if(itopNext.nele)
{TWGenMVs(); jrcLOOP igwLOOP if(RCG.twgmvs != RCG.gmvs) dif =
TRUE;}
SetMarginCtlW(FALSE, TRUE); TWGenitopNext(0); if(itopNext.nele)
{TWGenMVs();  jrcLOOP  igwLOOP  if(rctwMVs[jrc]  >  RCG.twgmvs)
rctwMVs[jrc]
=   RCG.twgmvs;   if(dolAct)   {dolHCb.SubFindtw(&dolFixRev);
if(dolHCb.Goods())
{dolHCb.GetSub(ig,jrc);   if(rctwMVs[dolCol]   >   RCG.twgmvs)
rctwMVs[dolCol]
= RCG.twgmvs;}}} SetMarginCtlB(); jrcLOOP {if(!IsDolCol(jrc)
&& GetColType(jrc) != -1) {rcMVa[jrc] = 0; if(GetColType(jrc) ==
1)
rctwMVs[jrc]  =  rcMVs[jrc]  =  0;}  if(IsDolCol(jrc)  &&  jrc  !=
dolCol)
{prec  junk  =  0,  junk2  =  0;  ConvDolToReal(jrc,  rcMVs[jrc],
rcMVa[jrc],
rcQuant [jrc]); ConvDolToReal(jrc, rctwMVs[jrc], junk, junk2);}}
ZEROOUT(rcPotMVs[0],     mrc);     ZEROOUT(rcPotMVa[0],     mrc);
XctMVPre(); for(i=0;
i<mrc;i++)  if(!dpNext.use[i])  {XctMVGet(i);  rcPotMVs[i]  =
rowFactor[i]
*  ctvMVs[i];  rcPotMVa[i]  =  rowFactor[i]  *  ctvMVa[i];}
XctMVPost();
SetMarginCtlW(TRUE,  FALSE);  for(jrc=0;jrc<nrc;jrc++)  {if(rcMVs
[jrc]
<  bigM)  rcMVs  [jrc]  *=  qOrgwtIn[jrc];  rcMVa  [jrc]  *=
qOrgwtIn[jrc];
```

```
if(rctwMVs [jrc] < bigM) rctwMVs[jrc] *= qOrgwtIn[jrc]; rctwMVa
[jrc]
*= qOrgwtIn[jrc]; rcQuant [jrc] *= qOrgwtOt[jrc]; rcQuantFix
[jrc]
*= qOrgwtOt[jrc];} return dif;} CKer::CKer() {Init();}
CKer::~CKer() {}
void CKer::Init() {mink = maxk = -1; nele = 0; sorted = FALSE;}
void CKer::Load(NEXTs& ns) {int i; Init(); LOOPs(ns,i)
{index[nele++] = i;
}} void CKer::Load(CKer& cks) {Init(); nele = cks.nele;
ARRAYCOPY(cks.index[0], index[0], nele);} int CKer::GetK(int
indexSearch)
{int k; for(k=0;k<nele;k++) if(index[k] == indexSearch) return
k;
return -1;} void CKer::GenMin() {int k; int inc = (sorted && 1 <
nele)
? nele - 1: 1; mink = -1; minv = prec_SORTMAX;
for(k=0;k<nele;k+=inc)
if(minv > val[k]) {minv = val[k]; mink = k;}} void
CKer::GenMax() {int k;
int inc = (sorted && 1 < nele) ? nele - 1: 1; maxk = -1;
maxv = -prec_SORTMAX; for(k=0;k<nele;k+=inc) if(maxv < val[k])
{maxv = val[k]; maxk = k;}} void CKer::SortAscend() {int kplace,
ktry;
int bestk; prec bestv; for(kplace=0; kplace < nele - 1;
kplace++)
{bestk = kplace; bestv = val[kplace]; for(ktry = kplace+1; ktry
< nele;
ktry++) if(val[ktry] < bestv) {bestk = ktry; bestv = val[ktry];}
Swap(index[bestk], index[kplace]); Swap(val [bestk], val
[kplace]);}
sorted = TRUE; if(mink != -1) {mink = 0; minv = val[mink];}
if(maxk != -1)
{maxk = nele-1; maxv = val[maxk];}} void CKer::SortDescend()
{int kplace,
ktry; int bestk; prec bestv; for(kplace=0; kplace < nele - 1;
kplace++)
{bestk = kplace; bestv = val[kplace]; for(ktry = kplace+1; ktry
< nele;
ktry++) if(val[ktry] > bestv) {bestk = ktry; bestv = val[ktry];}
Swap(index[bestk], index[kplace]); Swap(val [bestk], val
[kplace]);}
sorted = TRUE; if(mink != -1) {mink = nele-1; minv = val[mink];}
if(maxk != -1) {maxk = 0; maxv = val[maxk];}}
void CKer::DeleteValso(int k) {nele--; if(!sorted) {index [k]
= index[nele]; val[k] = val [nele];} else {ARRAYCOPY(index[k+1],
index[k],
nele - k); ARRAYCOPY(val [k+1], val [k], nele - k);} if(mink <
k) {}
else if(k < mink) {if(sorted) mink--; else if(mink == nele) mink
= k;}
```

```
else {GenMin();} if(maxk < k) {} else if(k < maxk) {if(sorted)
maxk--;
else if(maxk == nele) maxk = k;} else {GenMax();} index[nele] =
-1;}
IMPLEMENT_SERIAL(CKer, CObject,1)
#define PM3d pm3d[igs][iga][jrc]
#define ckMin ckmin[jrc]
#define ckMax ckmax[jrc]
CKer                PairMan::ckref[RCDTMAX_NRC];                int
PairMan::assigtok[CORTMAX_M1]
[RCDTMAX_NRC];     int     PairMan::pairMinCt     =     0;     prec
PairMan::pairMinInc = 0.0f;
int PairMan::pairMaxCt = 0; prec PairMan::blockMinInc = 0.0f;
int PairMan::blockMaxCt = 0; prec PairMan::walkMinInc = 0.0f;
int  PairMan::GetK(CKer&  ck,  int  ig,  int  jrc)  {int  k  =
assigtok[ig][jrc];
if(ck.index[k] != ig) k = ck.GetK(ig); return k;}
BOOL PairMan::IsProfitable(prec oprofit, prec& minInc)
{prec  cprofit  =  CTGetProfit();  cprofit  +=  0.01f;  oprofit  +=
0.01f;
if(IsEqual(cprofit, oprofit) || cprofit/oprofit < 1.0f + minInc)
return  FALSE;  else  return  TRUE;}  void  PairMan::BlockFin()
{if(!blockDone)
{blockDone    =    TRUE;    blockDoneProfit    =    CTGetProfit();
blockDoneCt++;}}
/*static*/     void     PairMan::LoadigPrep()     {int     jrc;
for(jrc=0;jrc<RCDTMAX_NRC;
jrc++) ckref[jrc].Init();} /*static*/ void PairMan::Loadig(int
jrc,
NEXTs&  igNext)  {int  ig,  k;  ckref[jrc].Load(igNext);  k  =  0;
LOOPs(igNext,
ig) assigtok[ig][jrc] = k++;}
/*static*/   void   PairMan::SetTolerance(int   prminCt,   prec
prminInc,
int prmaxCt, prec bkminInc, int bkmaxCt, prec wkminInc)
{pairMinCt  =  prminCt;  pairMinInc  =  prminInc;  pairMaxCt  =
prmaxCt;
blockMinInc  =  bkminInc;  blockMaxCt  =  bkmaxCt;  walkMinInc  =
wkminInc;}
/*static*/   void   PairMan::GetTolerance(int&   prminCt,   prec&
prminInc,
int& prmaxCt, prec& bkminInc, int& bkmaxCt, prec& wkminInc)
{prminCt  =  pairMinCt;  prminInc  =  pairMinInc;  prmaxCt  =
pairMaxCt;
bkminInc  =  blockMinInc;  bkmaxCt  =  blockMaxCt;  wkminInc  =
walkMinInc;}
void PairMan::PermBlockInit() {int ig, jrc; if(!permBlockClear)
{ZEROOUTSTRUCT(permBlock);          for(ig=0;ig<CORTMAX_M1;ig++)
for(jrc=0;
jrc<RCDTMAX_NRC;jrc++) permBlock[ig][ig][jrc] = TRUE;
permBlockClear  =  TRUE;}}  PairMan::PairMan()  {permBlockClear  =
FALSE;
```

```
PermBlockInit();} void PairMan::WalkInit()
{walkProfitBaseNote = CTGetProfit(); blockDoneCt = 0;}
void PairMan::Clear(NEXTs& jrcentertainSetNext) {int igs, iga,
jrc;
int     kigs,     kiga;     LOOPs(jrcentertainSetNext,     jrc)
ckLOOPk(ckref[jrc], kigs,
igs) ckLOOPk(ckref[jrc],kiga, iga) {PM3d.count = 0;
PM3d.blocked = permBlock[igs][iga][jrc];} blockDone = FALSE;
NextCopy(jrcentertainSetNext, jrcBestNext); for(jrc=0;
jrc<jrcentertainSetNext.mele;jrc++)              {ckMin.Init();
ckMax.Init();}}
void        PairMan::ItPrep()        {ZEROOUT(jrcBestNext.val[0],
jrcBestNext.mele);
ZEROOUT(bestEvalPend[0], jrcBestNext.mele);}
void PairMan::ItPrepCol(int jrc)
{if(ckMin.sorted     ||     ckMin.nele     !=     ckref[jrc].nele)
ckMin.Load(ckref[jrc]);
if(ckMax.sorted     ||     ckMax.nele     !=     ckref[jrc].nele)
ckMax.Load(ckref[jrc]);
itjrc = jrc; itks = itka = 0; ckMin.minv = prec_SORTMAX;
ckMax.maxv = - prec_SORTMAX; bestEvalPend[jrc] = TRUE;}
void PairMan::ItNoteSub(prec value) {int jrc = itjrc;
if(ckMin.minv > value) {ckMin.minv = value; ckMin.mink = itks;}
ckMin.val[itks++] = value;} void PairMan::ItNoteAdd(prec value)
{int jrc = itjrc; if(ckMax.maxv < value) {ckMax.maxv = value;
ckMax.maxk = itka;} ckMax.val[itka++] = value;}
void PairMan::ItDelSub(int ig, int jrc) {int k = GetK(ckMin, ig,
jrc);
if(k==-1)   return;   ckMin.DeleteValso(k);   bestEvalPend[jrc]   =
TRUE;}
void PairMan::ItDelAdd(int ig, int jrc) {int k = GetK(ckMax, ig,
jrc);
if(k==-1)   return;   ckMax.DeleteValso(k);   bestEvalPend[jrc]   =
TRUE;}

void PairMan::ItUpdateValueSub(int ig, int jrc, prec newValue)
{int   k   =   GetK(ckMin,   ig,   jrc);   ckMin.val[k]   =   newValue;
if(!ckMin.sorted)
ckMin.GenMin();   else   ckMin.SortAscend();   bestEvalPend[jrc]   =
TRUE;}
void PairMan::ItUpdateValueAdd(int ig, int jrc, prec newValue)
{int   k   =   GetK(ckMax,   ig,   jrc);   ckMax.val[k]   =   newValue;
if(!ckMax.sorted)
ckMax.GenMax();   else   ckMax.SortDescend();   bestEvalPend[jrc]   =
TRUE;}
void PairMan::GetBestPair(int& getigs, int& getiga, int& getjrc,
prec &getvalue) {GetBestPair(getigs, getiga, getjrc); if(getigs
!= -1)
getvalue = jrcBestNext.val[getjrc]; else getvalue = 0;}
void PairMan::GetBestPair(int& getigs, int& getiga, int& getjrc)
{int igs,
```

```
iga; int jrc; int bestjrc = -1; prec bestv = 0.0f;
LOOPs(jrcBestNext,jrc)
{if(bestEvalPend[jrc]) {if(!ckMax.nele || !ckMin.nele)
{bestEvalPend[jrc]
= FALSE; jrcBestNext.val[jrc] = -1.0f; continue;}
if(bestv < ckMax.maxv - ckMin.minv) {igs =
ckMin.index[ckMin.mink];
iga = ckMax.index[ckMax.maxk]; if(!PM3d.blocked) {bestigs[jrc] =
igs;
bestiga[jrc] = iga; bestv = jrcBestNext.val[jrc]
= ckMax.maxv - ckMin.minv; bestjrc = jrc; bestEvalPend[jrc] =
FALSE;
continue;} else {int ks, ka; if(!ckMin.sorted)
{ckMin.SortAscend();
ckMax.SortDescend();} for(ks=0;ks<ckMin.nele;ks++)
if(bestv < ckMax.maxv - ckMin.val[ks])
{for(ka=0;ka<ckMax.nele;ka++)
if(bestv < ckMax.val[ka] - ckMin.val[ks]) {igs =
ckMin.index[ks];
iga = ckMax.index[ka]; if(!PM3d.blocked) {bestigs[jrc] = igs;
bestiga[jrc]
= iga; bestv = jrcBestNext.val[jrc] = ckMax.val[ka] -
ckMin.val[ks];
bestjrc = jrc; bestEvalPend[jrc] = FALSE; break;}} else break;}
else break;}}} else {if(bestv < jrcBestNext.val[jrc])
{bestv = jrcBestNext.val[jrc]; bestjrc = jrc;}}} if(bestjrc != -
1)
{getjrc = bestjrc; getigs = bestigs[bestjrc]; getiga =
bestiga[bestjrc];}
else getigs = -1;} void PairMan::PrePair()
{pairProfitBaseNote = CTGetProfit();} int
PairMan::IncPairCount(int igs,
int iga, int jrc) {return ++PM3d.count;}
int PairMan::GetPairCount(int igs, int iga, int jrc) {return
PM3d.count;}
void PairMan::PostPair(int igs, int iga, int jrc) {PM3d.count++;
if (pairMinCt <= PM3d.count)
if(!IsProfitable(pairProfitBaseNote,
pairMinInc) || PM3d.count == pairMaxCt) BlockPair(igs, iga,
jrc);}
void PairMan::BlockPair(int igs, int iga, int jrc) {PM3d.blocked
= TRUE;
bestEvalPend[jrc] = TRUE; BlockFin();} void
PairMan::BlockPair(int igs,
int iga, int jrc, prec curProfit) {PM3d.blocked = TRUE;
bestEvalPend[jrc]
= TRUE; if(!blockDone) {BlockFin(); blockDoneProfit =
curProfit;}}
void PairMan::Blockigs(int iga, int jrc) {int igs;
ckLOOP(ckref[jrc], igs)
PM3d.blocked = TRUE; bestEvalPend[jrc] = TRUE; BlockFin();}
```

```
void   PairMan::Blockiga(int   igs,   int   jrc)   {int   iga;
ckLOOP(ckref[jrc], iga)
PM3d.blocked = TRUE; bestEvalPend[jrc] = TRUE; BlockFin();}
void   PairMan::PermBlockPair(int   igs,   int   iga,   int   jrc)
{BlockPair(igs,
iga, jrc); permBlock[igs][iga][jrc] = TRUE; permBlockClear =
FALSE;}
BOOL PairMan::GetPermBlockPair(int igs, int iga, int jrc)
{return            permBlock[igs][iga][jrc];}            BOOL
PairMan::ProfitableSinceBlocked()
{BOOL rt; if(!blockDone || blockDoneCt == blockMaxCt) rt =
FALSE;
else rt= IsProfitable(blockDoneProfit, blockMinInc); return rt;}
BOOL            PairMan::WalkProfitable()            {return
IsProfitable(walkProfitBaseNote,
walkMinInc);} void PairMan::SetTemp(int setigs, int setiga, int
setjrc)
{igsTemp = setigs; igaTemp = setiga; jrcTemp = setjrc;}
void PairMan::GetTemp(int& getigs, int& getiga, int& getjrc)
{getigs = igsTemp; getiga = igaTemp; getjrc = jrcTemp;}
IMPLEMENT_SERIAL(PairMan, CObject,1) RCstor::RCstor() {level = -
1;}
#define storArray(anchor, elements) ARRAYCOPY(::anchor, anchor,
elements)
#define storArray2d(anchor, maxi, maxj) ARRAYCOPY2d(::anchor,
maxi, maxj,\
anchor)
#define storScalar(scalar)\
scalar = ::scalar
#define storStruct(structx) STRUCTCOPY(::structx,structx)
#define storRC(RC) {memcpy((void*)&(RC), (void*) &(::RC),\
sizeof(resConduitRCstor));}
void   RCstor::Out(int   setlevel/*=0*/)   {level   =   setlevel;
if(0<=level)
{int ig, jrc; ctImage.Out(2); for(jrc=0;jrc<nrc;jrc++) igbLOOP
{rc[ig]
[jrc].allotment          =          ::rc[ig][jrc].allotment;}
storArray(potential[0],mrc);}
if(1<=level)      {storArray(qOrg[0],      nrc);}}      void
RCstor::OutProfit()
{profit = CTGetProfit();}
#undef storArray
#undef storArray2d
#undef storScalar
#undef storStruct
#undef storRC
#define storArray(anchor, elements) ARRAYCOPY(anchor, ::anchor,
elements)
#define storArray2d(anchor, maxi, maxj) ARRAYCOPY2d(anchor,
maxi, maxj,\
::anchor)
#define storScalar(scalar)\
```

```
::scalar = scalar
#define storStruct(structx) STRUCTCOPY(structx,::structx)
#define storRC(RC) {memcpy((void*)&(::RC), (void*) &(RC),\
sizeof(resConduitRCstor));}
void RCstor::In() {if(0<=level) {int ig, jrc; ctImage.In();
for(jrc=0;
jrc<nrc;jrc++) igbLOOP {::rc[ig][jrc].allotment =
rc[ig][jrc].allotment;
GenGroupFactor(ig, jrc, FALSE);} storArray(potential[0],mrc);}
if(1<=level) {storArray(qOrg[0], nrc);} if(level==0) {int i;
for(i=0;
i<mrc;i++) {ffp[i].potential = potential[i]; GenRowFactor(i);}}
else SetMarginCtlW(TRUE, FALSE); GenNextMV(jrcNext);}
prec RCstor::GetProfit() {return profit;}
#undef storArray
#undef storArray2d
#undef storScalar
#undef storStruct
#undef storRC
IMPLEMENT_SERIAL(RCstor, CObject,1) RCstorp::RCstorp()
{pRCstor = new RCstor;} RCstorp::~RCstorp() {JDELETE(pRCstor);}
void RCstorp::Out(int setlevel/*=0*/) {pRCstor->Out(setlevel);}
void RCstorp::OutProfit() {pRCstor->OutProfit();} void
RCstorp::In()
{pRCstor->In();} prec RCstorp::GetProfit() {return pRCstor-
>GetProfit();}
IMPLEMENT_SERIAL(RCstorp, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]rcfilter.h[~~~~~~~~~~~~~~~~~~~~
~~~~~~~~
#ifndef RCFilter_h
class RCFilter : public CObject {DECLARE_SERIAL(RCFilter);
public: rcType tt[4]; int nele; BOOL allPass; public:
RCFilter();
~RCFilter(); void Init(); void Include(rcType include); void
SetAll();
void SetButFix(); BOOL Pass(ResConduit& rc);
static CString GetType(ResConduit& rc); BOOL AllPass();
void Serialize(CArchive& ar);};
#define RCFilter_h
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]rcfilter.cpp[~~~~~~~~~~~~~~~~~~
~~~~~~~~~~
#include "stdafx.h"
#include "jtools.h"
#include "Cort.h"
#include "RCDT.h"
#include "RCFilter.h"
RCFilter::RCFilter() {Init();} RCFilter::~RCFilter() {}
void RCFilter::Init() {nele = 0; allPass = FALSE;}
void RCFilter::Include(rcType include) {tt[nele++] = include;}
void RCFilter::SetAll() {Init(); Include(rctNor);
Include(rctFix);
```

```
Include(rctBas); Include(rctVar); allPass = TRUE;}
void    RCFilter::SetButFix()    {Init();    Include(rctNor);
Include(rctBas);
Include(rctVar);} BOOL RCFilter::Pass(ResConduit& rc) {for(int
i=0;
i<nele;i++) {if(tt[i] == rc.type) return TRUE;} return FALSE;}
/*static*/ CString RCFilter::GetType(ResConduit& rc)
{if(rc.type == rctNor) return "rctNor"; if(rc.type == rctFix)
return "rctFix"; if(rc.type == rctBas) return "rctBas";
if(rc.type == rctVar) return "rctVar"; return "Unknown";}
BOOL RCFilter::AllPass() {return allPass;}
void RCFilter::Serialize(CArchive& ar) {CObject::Serialize(ar);}
IMPLEMENT_SERIAL(RCFilter, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]resconduit.h[~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~
#ifndef resConduit_h
enum rcType {rctEmpty=0,rctNor=1,rctFix=2,rctBas=3,rctVar=4};
class ResConduit : public CObject {DECLARE_SERIAL(ResConduit);
public: prec allotment; prec factor; prec* pFactorDep; BOOL
onCorner;
int  ir;  prec  dedrs;  prec  dedra;  prec  dedrsBfPot;  prec
dedraBfPot;
prec gmvs; prec gmva; prec emvs; prec emva; int subBlk; int
subBlkHold;
rcType type; int nir; BOOL dirPut; int igUp; int ieDown; int id;
int rSet;
prec twgmvs; int twuig; int twujrc; int twuDownig; prec twQuant;
prec twFactor; prec twMQuant; long twFootPrt; prec twgmva;
prec  rstop[RCDTMAX_PT];  prec  estop[RCDTMAX_PT];  prec  dedr
[RCDTMAX_PT];
BOOL   dedrInfinite[RCDTMAX_PT];   public:   ResConduit();
~ResConduit();
void OrientRegularatoe(); BOOL IsConvex(); BOOL IsMCsInfinite();
public: void Serialize(CArchive& ar);};
#define resConduit_h
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]resconduit.cpp[~~~~~~~~~~~~~~~~
~~~~~~~~~~~~
#include "stdafx.h"
#include "jtools.h"
#include "ResConduit.h"
ResConduit::ResConduit() {} ResConduit::~ResConduit() {}
void ResConduit::OrientRegularatoe() {prec wrstop[DIM1(rstop)];
prec westop[DIM1(rstop)]; prec wdedr [DIM1(rstop)]; int i, n =
0;
if(ZeroPress(rstop[0]) || ZeroPress(estop[0])) {wrstop[0] = 0;
westop[0]
= 0; n++;} for(i=0;i<nir+1;i++) {wrstop[n] = rstop[i]; westop[n]
= estop[i]; n++;} ZEROOUT(dedrInfinite[0], n); if(!wrstop[1])
{wrstop[1]
=   0.0001;   dedrInfinite[0]   =   TRUE;}   for(i=0;i<n-1;i++)
if(wrstop[i]
```

```
>= wrstop[i+1] - TOLERANCE) {wrstop[i+1] = wrstop[i] * 1.0001;
dedrInfinite[i] = TRUE;} if(!westop[1]) {westop[1] = 0.0001;
dedrInfinite[0] = TRUE;} for(i=0;i<n-1;i++) if(westop[i] >=
westop[i+1]
- TOLERANCE) westop[i+1] = westop[i] * 1.0001; for(i=0; i<n-1;
i++)
{wdedr[i] = (westop[i+1] - westop[i])/(wrstop[i+1] - wrstop[i]);
if(1 < westop[i+1]) {n = i+1; break;}} nir = n-1;
BOUNDP(estop[nir],1);
wdedr [nir] = 0; dedrInfinite[nir] = FALSE; for(i=0; FALSE &&
i<=nir; i++)
{if(i) {} CString ss;} ARRAYCOPY(wrstop[0], rstop[0], nir+1);
ARRAYCOPY(westop[0], estop[0], nir+1); ARRAYCOPY(wdedr [0],
dedr[0],
nir+1);} BOOL ResConduit::IsConvex() {for(int ii = 0; ii < nir -
2; ii ++)
if(dedr[ii] < dedr[ii+1]) return TRUE; return FALSE;}
BOOL ResConduit::IsMCsInfinite() {if(!ir && onCorner) return
TRUE;
if(!onCorner) return dedrInfinite[ir]; else return
dedrInfinite[ir-1];}
void ResConduit::Serialize(CArchive& ar)
{CObject::Serialize(ar);}
IMPLEMENT_SERIAL(ResConduit, CObject,1)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]wed.h[~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
#ifndef WED_H
class DSCS; void WedDistInit(); void WedLoadCheck(DCSpec& dc);
void WedDistGen(); void WMInit(); void WMNoteScenario();
int WedGenBase(int step); void WedMapAppend0();
void WedMapAppend1(prec v0, prec v1, prec v2);
void WedMapAppend2(Locator& llL, JCellM* pMap);
void GetAverageCost(DSCS& dlg, prec profitSup0, prec profitSupX,
int iProdSupply, prec quantX, prec priceX, prec& aCost);
int WedGenSupply(int step); void WedGenDemandPt(prec quant, int
iPot,
int jRC, WedMeaner& wm); int WedGenDemand(int step);
void WedStep1(int (*pstepExecuteSet)(int), WView* pView);
UINT WedStep1Do(LPVOID); void WedStep2();
#define WED_H
#endif
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~]wed.cpp[~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
#include "stdafx.h"
#include <string.h>
#include "jtools.h"
#include "cort.h"
#include "rcdt.h"
#include "dol.h"
#include "colrowid.h"
#include "Locator.h"
#include "wed.h"
```

```
CWinThread* pstepThread = NULL; WView* pstepView; int
(*pstepExecute)(int);
prec lagTime;
#define FindAss(struct_, field, value, index, maxIndex)
for(index=0;\
index<maxIndex;index++) if(struct_[index].field == value) break;
WDoc* pDoc=NULL; Locator llD, llR, llP, llF; Locator coln, colc;
struct ftDefstruct {char* rowTitle; ROWNAME rowName; int iRowCt;
char* format; char* rowDefault; BOOL substanceEdit; int
junkRowCount;}
ftDef[] ={{"filler", rowFBlank, -1, "ff", "", FALSE, -1},
{"Internal Producer's Surplus", rowFIPS, 1, "f$2",
"", FALSE, 1}, {"Standard Error", rowFIPSse, 2,
"ff2", "", FALSE, 2}, {"", rowFBlank, -1, "ff", "", FALSE, 3},
{"Change in wCash", rowFDCash, 4, "f$2", "", FALSE, 4},
{"Standard Error", rowFDCashse, 5, "ff2", "", FALSE, 5}, {"",
rowFBlank,
-1, "ff", "", FALSE, 6}, {"wCash (Beginning)", rowFCash, 7,
"f$2", "",
FALSE, 7}, {"Marginal Value", rowFCashMV, 8, "f$2", "", FALSE,
8},
{"Standard Error", rowFCashMVse, 9, "ff2", "", FALSE, 9}, {"",
rowFBlank, -1, "ff", "", FALSE, 10}, {"Sum wFilV", rowFSFValue,
11,
"f$2", "", FALSE, 11}, {"Standard Error", rowFSFValuese, 12,
"ff2", "",
FALSE, 12}, {"", rowFBlank, -1, "ff", "", FALSE, 13}, {"Sum
wWTMD",
rowFSWTDM, 14, "f$2", "", FALSE, 14}, {"Standard Error",
rowFSWTDMse, 15,
"ff2", "", FALSE, 15}, {"", rowFBlank, -1, "ff", "", FALSE,
16},
{"Parameters", rowFPara, -1, "ff", "", FALSE, 17},
{"__Allocation",
rowFAType, 18, "el", "Direct", TRUE, 18}, {"__Maximization",
rowFMax, 19,
"el", "IPS", TRUE, 19}, {"__wCash Type", rowFCType, 20, "el",
"Spread Out", TRUE, 20}, {"__Rand Seed", rowFRSeed, 21, "ei",
"1", TRUE,
21}, {"__N Sample", rowFNSample, 22, "ei", "35", TRUE, 22},
{"__MC/MV Display", rowFMCDisplay, 23, "el", "Infinite Series",
TRUE, 23}
, {"__Base", rowFBlank, -1, "ff", "", FALSE, 24},
{"____Max RW Iterations", rowFRWBaseIter, 25, "ei", "3", TRUE,
25},
{"____Max RW Time (sec)", rowFRWBaseTime, 26, "ef3", "20", TRUE,
26},
{"__Case", rowFBlank, -1, "ff", "", FALSE, 27},
{"____Max RW Iterations", rowFRWCaseIter, 28, "ei", "3", FALSE,
28},
{"____Max RW Time (sec)", rowFRWCaseTime, 29, "ef3", "20",
FALSE, 29} }
```

```
; long rndMasterSeed = 23556; int nSample = 84;
BOOL allocDirect = FALSE; BOOL baseCurrent = FALSE;
struct { int wtProw0; Dist* pDist; int rcdtRow; }
prod[jcellRowsMax]; int mProdw; struct { int wtRrow0; int
cortRow;
} resSimple[jcellRowsMax]; int mRS; int mCTw; int
rsiCashRow;
WedMeaner gWM[jcellRowsMax]; class ResGroup { public: int
wtRrow0;
int wtRrow1; int rcdtCol; NEXTs ieList; WedMeaner* pgWM; };
ResGroup resGroup[jcellRowsMax]; int mRG; int nRCw;
WedMeaner cWM[CORTMAX_N]; prec sumPayOt [CORTMAX_N];
prec wtmdProd [CORTMAX_N]; prec priceRaw [CORTMAX_N];
prec valueFil [CORTMAX_N]; prec priceExe [CORTMAX_N];
WedMeaner bWM[CORTMAX_M1]; prec bQt[CORTMAX_M1]; WedMeaner
ipspWM;
WedMeaner cashWM; WedMeaner fillWM; WedMeaner wtmdWM;
WedMeaner profitBaseWM; WedMeaner* pProfitWM;
WedMeaner rcWM[RCDTMAX_NRC]; prec rcQt[RCDTMAX_NRC];
WedMeaner potWM[CORTMAX_M1]; RCstor** pBase = NULL;
RCstor** pWork = NULL; int nBase = 0; int nWork = 0;
RCstor* pMean = NULL; RCstor* pHold = NULL; int sizeBaseWork;
NEXTs jrcNextWol;
#define wolAct jrcNextWol.nele
long wolRow; long wolCol; prec wolPrice[NEXTMAX]; prec wolQuant;
prec wolQuantColFixBuy; int wolIncn; void WedDistInit()
{/*initialize distributions using rndMasterSeed as a seed */}
void WedLoadCheck(DCSpec& dc) {static BOOL reloading = FALSE;
int i;
int iProd; int iRS; int iRG; mProdw = 0; LOCLOOP0(llP)
{prod[mProdw]
.wtProw0 = llP.ir0; llD.FindNote(0, colDName, llP.p0(colPDist)-
>rdString);
prod[mProdw].pDist = llD.p0Map(colDDist)->pDist; prod[mProdw]
.rcdtRow = mProdw; mProdw++;} BOOL resUsed[jcellRowsMax] =
{FALSE};
LOCLOOP0(llP) LOCLOOP1(llP)
{i = llR.pwA->ResourceFindRow(llP.p1(colPResource)->rdString);
resUsed[i]
= TRUE;} llR.pwA->GeniCashRow(); resUsed[iCashRow] = TRUE; mRS =
0;
mCTw = mProdw; LOCLOOP0(llR) if(resUsed[llR.ir0]
&& llR.p0(colRAvailability)->IsString("Fixed", "FixBuy"))
{resSimple[mRS]
.wtRrow0 = llR.ir0; resSimple[mRS].cortRow = mCTw++; mRS++;}
FindAss(resSimple,wtRrow0,iCashRow, rsiCashRow, mRS);
NextPrep(jrcNextWol,
RCDTMAX_NRC); mRG = 0; nRCw = 0; wolQuantColFixBuy = 0;
LOCLOOP0(llR)
{int nRCwinc = 0; BOOL buyable = FALSE; LOCLOOP1(llR)
if(resUsed[llR.ir1])
```

```
{resGroup[mRG].wtRrow0 = llR.ir0; resGroup[mRG].wtRrow1 =
llR.irl;
resGroup[mRG].rcdtCol = nRCw; NextPrep(resGroup[mRG].ieList,
mCTw); mRG++;
if(!nRCwinc) {if(llR.p0(colRAvailability)->IsString("Buyable"))
{NextInsert(jrcNextWol, nRCw); wolPrice[nRCw] =
llR.p0(colRPayPrice)
->VFetch(); buyable = TRUE;} if(llR.p0(colRAvailability)
->IsString("FixBuy")) {wolQuantColFixBuy +=
llR.p0(colRPayPrice)->VFetch()
* llR.p0(colRQuanity)->VFetch() ;}} nRCwinc = 1;} nRCw +=
nRCwinc;}
if(wolAct) {jrcNextWol.Resetmele(nRCw + 1); wolRow =
resSimple[rsiCashRow]
.cortRow; wolCol = nRCw; wolIncn = 1;} else {wolRow = -1; wolCol
= -1;
wolIncn = 0;} wolQuant = llR.DGet(iCashRow, colRQuanity)-
>VFetch();
if(CORTMAX_N < mProdw) dc.Note("Too many products.", wtP);
else if(CORTMAX_M1 - 1 < mCTw) dc.Note("Too many products and
resources.",
wtP); else if(RCDTMAX_NRC < nRCw + wolIncn)
dc.Note("Too many group resources.", wtR); if(dc.IsError())
return;
CortPrep(mCTw, mProdw, mProdw); RCDTPrep(mCTw, nRCw+ wolIncn,
mProdw);
for(iProd=0;iProd<mProdw;iProd++) {CTLoadaElement(iProd, iProd,
1);
llP.NoteIr(prod[iProd].wtProw0); prec coefCash = 0.0;
LOCLOOP1(llP)
{int iRes = llR.pwA->ResourceFindRow(llP.p1(colPResource)-
>rdString);
llR.NoteIr(iRes); if(llR.pwA->pSide[iRes]->irowType == 0) {int
iRS;
prec coef = llP.p1(colPQuantity)->VFetch();
if(llR.p0(colRAvailability)
->IsString("Fixed", "FixBuy")) {FindAss(resSimple, wtRrow0,
iRes, iRS, mRS)
; CTLoadaElement(resSimple[iRS].cortRow, iProd, coef);}
else {coef *= llR.p0(colRPayPrice)->VFetch();
if(coln.DGet(rowFCType)
->IsString("Fold In")) coef *= llP.p1(colPDtoCash)->VFetch();
coefCash += coef;}} else {FindAss(resGroup, wtRrow1, iRes, iRG,
mRG);
NextInsert(resGroup[iRG].ieList, iProd);}}
if(ZeroPress(coefCash))
CTLoadaElement(resSimple[rsiCashRow].cortRow, iProd, coefCash);}
BOOL resSimpleDPLoaded[jcellRowsMax] = {FALSE}; for(iRG=0;
iRG<mRG; iRG++)
{ResConduit rc; ZEROOUTSTRUCT(rc);
llR.NoteIr(resGroup[iRG].wtRrow1);
JCellM* pMap = llR.p1Map(colRGEffectiveness); MAPi {rc.rstop[i]
```

```
= pMAPj0->VFetch(); rc.estop[i] = pMAPj1->VFetch()/100.0; rc.nir
= i;}
rc.OrientRegularatoe();          LoadGroup(resGroup[iRG].ieList,
resGroup[iRG]
.rcdtCol, rc); if(llR.p1(colRGDedicate)->IsString("Yes"))
{prec      qt       =       llR.p1(colRGAllot)->VFetch();
SetrcTypeAsFix(resGroup[iRG]
.ieList.lo,          resGroup[iRG].rcdtCol,          qt);}
for(iRS=0;iRS<mRS;iRS++)
if(resSimple[iRS].wtRrow0 == resGroup[iRG]
.wtRrow0    &&    !resSimpleDPLoaded[iRS])    {NEXTs    nextDP;
NextPrep(nextDP, mCTw);
NextInsert(nextDP, resSimple[iRS].cortRow); LoadGroupDP(nextDP,
resGroup[iRG].rcdtCol);   resSimpleDPLoaded[iRS]   =   TRUE;}}
for(iProd=0;
iProd<mProdw;iProd++) {sumPayOt[iProd] = 0; wtmdProd[iProd] = 0;
llP.NoteIr(prod[iProd].wtProw0); LOCLOOP1(llP)
{int   iRes   =   llR.pwA->ResourceFindRow(llP.p1(colPResource)-
>rdString);
llR.NoteIr(iRes); if(llR.pwA->pSide[iRes]->irowType == 0)
{if(llR.p0(colRAvailability)->IsString("Buyable"))
sumPayOt[iProd]
+=   llP.p1(colPQuantity)->VFetch()   *   llR.p0(colRPayPrice)-
>VFetch();
else {if(coln.DGet(rowFMax)->IsString("IPS")) wtmdProd[iProd]
+=    llP.p1(colPQuantity)->VFetch()    *    llR.p0(colRWTMD)-
>VFetch();}}}
priceRaw[iProd]          =          llP.p0(colPPrice)->VFetch();
if(coln.DGet(rowFMax)
->IsString("IPS"))  {valueFil[iProd]  =  llP.p0(colPFillValue)-
>VFetch();
priceExe[iProd]   =   priceRaw[iProd]   +   valueFil[iProd]   -
sumPayOt[iProd]
- wtmdProd[iProd];} else {valueFil[iProd] = 0; priceExe[iProd]
= priceRaw[iProd] - sumPayOt[iProd];}} if(coln.DGet(rowFMax)
->IsString("IPS"))  pProfitWM  =  &ipspWM;  else  pProfitWM  =
&cashWM;
SPREAD(bQt,      mCTw,      -1);      for(iRS=0;iRS<mRS;iRS++)
bQt[resSimple[iRS]
.cortRow]  =  llR.DGet(resSimple[iRS].wtRrow0,  colRQuanity)-
>VFetch();
for(i=mProdw;i<mCTw;i++)  {SPURTP(bQt[i],  0);}  WedDistInit();
for(i=0;
i<nSample;i++)  WedDistGen();  for(i=0;i<mProdw;i++)  bQt[i]  /=
nSample;
for(iRG=0; iRG<mRG; iRG++) rcQt[resGroup[iRG].rcdtCol]
=    llR.DGet(resGroup[iRG].wtRrow0,    colRQuanity)->VFetch();
if(wolAct)
{bQt[wolRow] = 0; rcQt[wolCol] = wolQuant - wolQuantColFixBuy;
DolLoadBody(jrcNextWol, wolRow, wolCol); wolPrice[wolCol] = 1.0;
DolLoadPrice(wolPrice);}                                    else
{bQt[resSimple[rsiCashRow].cortRow]
```

```
= wolQuant - wolQuantColFixBuy;} CTLoadb(bQt);
CTLoadc(priceExe);
LoadqOrg(rcQt); LoadPotential(bQt); WMInit();
try {pBase = new RCstor*[nSample]; pWork = new RCstor*[nSample];
nBase = 0; nWork = 0; for(i=0;i<nSample;i++) {pBase[nBase++] =
new RCstor;
pWork[nWork++] = new RCstor;} pMean = new RCstor; pHold = new
RCstor;}
catch(...) {dc.Note("Out of memory.", coln.DGet(rowFNSample));
dc.Note(wtF); return;}} void WedDistGen() {int iProd;
LOCLOOP0(llD)
{Dist* pDist = (llD.p0Map(colDDist))->pDist;
if(coln.DGet(rowFAType)
->IsString("Indirect")) pDist->GenRndQ(); else pDist-
>GenMeanQ();
SPURTP(pDist->rndQuant, 0); pDist->meaner.Note(pDist-
>rndQuant);}
for(iProd=0;iProd<mProdw;iProd++)
{llP.NoteIr(prod[iProd].wtProw0);
prec qt = prod[iProd].pDist->rndQuant * llP.p0(colPDistPC)-
>VFetch()
/ 100.0; potWM[iProd].q.Note(qt); bQt[iProd] += qt;}} void
WMInit()
{int i, j; for(i=0; i<mRG; i++) {resGroup[i].pgWM = &gWM[i];
resGroup[i]
.pgWM->Init();} for(i=0;i<mProdw;i++) cWM[i].Init();
for(i=0;i<mCTw;i++)
{bWM[i].Init(); potWM[i].Init();} ipspWM.Init(); fillWM.Init();
cashWM.Init(); wtmdWM.Init(); for(j=0;j<nRCw+wolIncn;j++)
rcWM[j].Init();}
void WMNoteScenario() {int i, j; int iRS; pHold->Out(1);
CTClearGetvh();
BOOL infMC = coln.DGet(rowFMCDisplay)->IsString("Infinite
Series");
if(!RCGetResultVec()) infMC = FALSE; for(i=0;i<mRG;i++)
{int ig = resGroup[i].ieList.lo; int jrc = resGroup[i].rcdtCol;
prec mv,
qt, junk; if(!infMC) GetGroupMV(ig, jrc, mv, junk, qt);
else GetGrouptwMV(ig, jrc, mv, junk, qt); if(!GetpGroup(ig,jrc)
->IsMCsInfinite()) resGroup[i].pgWM->v.Note(mv); else
resGroup[i]
.pgWM->v.NoteInfinite(); resGroup[i].pgWM->q.Note(qt);} for(i=0;
i<mCTw;
i++) {CTGetvData(i); bWM[i].v.Note(ctvMVs[i]);
bWM[i].q.Note(ctvQuant[i]);
} prec ipsp = 0; prec cash = 0; prec fill = 0; prec wtmd = 0;
for(j=0;
j<mProdw;j++) {CTGethQuant(j); if(ZeroPress(cthQuant[j]))
{cWM[j]
.v.Note(priceRaw[j] + valueFil[j]);} else if
(ZeroPress(CTGetbOrg(j)))
{CTGethData(j,3, mProdw); cWM[j].v.Note(cthMC[j] + sumPayOt[j]
```

```
+ wtmdProd[j]);} else {CTGethData(j,3, mProdw); prec mc =
cthMC[j]
+ sumPayOt[j] + wtmdProd[j]; int rtCond; prec rcMC;
RCGetRowMC(j, infMC,
rtCond, rcMC); if(rtCond == 0) {cWM[j].v.Note(mc + rcMC);} else
cWM[j]
.v.NoteInfinite();} cWM[j].q.Note(cthQuant[j]); ipsp +=
cthQuant[j]
* (priceRaw[j] + valueFil[j] - sumPayOt[j] - wtmdProd[j]);
cash += cthQuant[j] * (priceRaw[j] - sumPayOt[j]); fill +=
cthQuant[j]
* valueFil[j];} for(iRS=0;iRS<mRS;iRS++) wtmd +=
ctvQuant[resSimple[iRS]
.cortRow] * llR.DGet(resSimple[iRS].wtRrow0, colRWTMD)-
>VFetch();
ipsp -= DolGetGroupPayOut(); cash -= DolGetGroupPayOut();
ipspWM.v.Note(ipsp); cashWM.v.Note(cash); fillWM.v.Note(fill);
wtmdWM.v.Note(wtmd); for(j=0; j<nRCw + wolIncn; j++) {prec val;
if(!infMC)
val = rcMVs[j]; else val = rctwMVs[j]; if(val < gmv_MAX) rcWM[j]
.v.Note(val); else rcWM[j].v.NoteInfinite();
rcWM[j].q.Note(rcQuant[j]);}
for(i=0;i<mProdw;i++) potWM[i].v.Note(rcPotMVs[i]);
for(i=0;i<mProdw;i++)
bQt[i] -= cthQuant[i]; for(i=0;i<mProdw;i++)
{bQt[prod[i].rcdtRow]
*= llP.DGet(prod[i].wtProw0, colPCOver)->VFetch()/100.0;
if(ZeroPress(bQt[prod[i].rcdtRow]) <= 0) bQt[prod[i].rcdtRow] =
0;}
pHold->In();} int WedGenBase(int step) {switch(step)
{case 1: {SetrwMax((long) coln.DGet(rowFRWBaseIter)->VFetch(),
coln.DGet(rowFRWBaseTime)->VFetch()); MaxFresh();
RCRoundAdjustmentwAW();
pMean->Out(1); WMInit(); WedDistInit(); for(int
iCase=0;iCase<nSample;
iCase++) {pMean->In(); WedDistGen(); MaxPotential(bQt);
RCRoundAdjustmentwAW(); pBase[iCase]->Out(1); WMNoteScenario();}
break;}
case 2: {int i,j; if(wolAct) {rcWM[nRCw + wolIncn-1]
.q.SetMeanShift(wolQuantColFixBuy);
bWM[resSimple[rsiCashRow].cortRow]
= rcWM[nRCw + wolIncn-1];} else
bWM[resSimple[rsiCashRow].cortRow]
.v.SetMeanShift(1.0); for(i=1;i<llF.pwA->mSide;i++) {FMSpec
fmSpec;
fmSpec = colc.DGet(i)->fmSpec; colc.DGet(i)-
>DataLoad(*coln.DGet(i),TRUE,
TRUE); colc.DGet(i)->fmSpec.cs = fmSpec.cs;} LOCLOOP0(llD)
{Dist* pDist = (llD.p0Map(colDDist))->pDist; pDist->sumMV = 0;
pDist->active = FALSE;} for(i=0;i<mProdw;i++)
{llP.NoteIr(prod[i].wtProw0)
; prod[i].pDist->sumMV += potWM[prod[i].rcdtRow].v.GetMean()
```

```
* llP.p0(colPDistPC)->VFetch()/100.0; prod[i].pDist->active =
TRUE;}
LOCLOOP0(llD) {Dist* pDist = (llD.p0Map(colDDist))->pDist;
if(pDist->active) {llD.p0(colDMean)->SetRawDataMean(pDist-
>meaner);
llD.p0(colDMV)->SetRawData(pDist->sumMV);}} for(i=0;i<mRS;i++)
{WedMeaner& wm = bWM[resSimple[i].cortRow];
llR.NoteIr(resSimple[i]
.wtRrow0); llR.p0(colRMeanUse)->SetRawDataMean(wm.q); if(i ==
rsiCashRow)
{colc.DGet(rowFCashMV)->SetRawDataMean(wm.v);
colc.DGet(rowFCashMVse)
->SetRawDataSE(wm.v);} llR.p0(colRMV)->SetRawDataMean(wm.v);}
for(i=0;
i<mRG;i++) {WedMeaner& wm = *(resGroup[i].pgWM);
llR.NoteIr(resGroup[i]
.wtRrow1); llR.p1(colRGMeanUse)->SetRawDataMean(wm.q);
llR.p1(colRGMV)
->SetRawDataMean(wm.v);} for(j=0;j<nRCw;j++) {WedMeaner& wm =
rcWM[j];
FindAss(resGroup, rcdtCol, j, i, mRG);
llR.NoteIr(resGroup[i].wtRrow0);
llR.p0(colRMeanUse)->SetRawDataMean(wm.q); llR.p0(colRMV)
->SetRawDataMean(wm.v);} if(wolAct) {colc.DGet(rowFCashMV)
->SetRawDataMean(rcWM[nRCw+wolIncn-1].v);
colc.DGet(rowFCashMVse)
->SetRawDataSE(rcWM[nRCw+wolIncn-1].v); llR.NoteIr(iCashRow);
llR.p0(colRMV)->SetRawDataMean(rcWM[nRCw+wolIncn-1].v);}
prec resUsed[jcellRowsMax] = {0}; for(i=0;i<mProdw;i++)
{llP.NoteIr(prod[i].wtProw0); LOCLOOP1(llP)
{int iRes = llR.pwA->ResourceFindRow(llP.p1(colPResource)-
>rdString);
llR.NoteIr(iRes); if(llR.pwA->pSide[iRes]->irowType == 0)
if(llR.p0(colRAvailability)->IsString("Buyable")) resUsed[iRes]
+= llP.p1(colPQuantity)->VFetch() * cWM[i].q.GetMean();}}
for(i=1;
i<llR.pwA->mSide;i++) if(ZeroPress(resUsed[i]) &&
!llR.DGet(i,colRMeanUse)
->rdSource) llR.DGet(i,colRMeanUse)->SetRawData(resUsed[i]);
for(i=0;
i<mProdw;i++) {llP.NoteIr(prod[i].wtProw0); int ij =
prod[i].rcdtRow;
llP.p0(colPMC)->SetRawDataMean(cWM[ij].v);
llP.p0(colPMeanDemand)
->SetRawDataMean(potWM[ij].q); llP.p0(colPMeanSupply)
->SetRawDataMean(cWM[ij].q);} colc.DGet(rowFIPS)-
>SetRawDataMean(ipspWM.v);
colc.DGet(rowFIPSse)->SetRawDataSE(ipspWM.v);
colc.DGet(rowFDCash)
->SetRawDataMean(cashWM.v); colc.DGet(rowFDCashse)-
>SetRawDataSE(cashWM.v);
```

```
colc.DGet(rowFCash)->DataLoad(*llR.pwA-
>DGet(resSimple[rsiCashRow]
.wtRrow0,          colRQuanity));          colc.DGet(rowFSFValue)-
>SetRawDataMean(fillWM.v);
colc.DGet(rowFSFValuese)->SetRawDataSE(fillWM.v);
colc.DGet(rowFSWTDM)
->SetRawDataMean(wtmdWM.v);                colc.DGet(rowFSWTDMse)-
>SetRawDataSE(wtmdWM.v);
baseCurrent   =   TRUE;   profitBaseWM   =   *pProfitWM;   pDoc-
>SurfaceWindow(wtF);
pDoc->GlobalRefresh();     break;}}     return     0;}     prec
wmaData[mapCellVerMax]
[mapCellHorMax]; int wmanGenRow; int wmanPutRow; JTimer wmaTime;
void   WedMapAppend0()   {wmanGenRow   =   0;   wmanPutRow   =   0;
wmaTime.Init();}
void    WedMapAppend1(prec    v0,    prec    v1,    prec    v2)
{wmaData[wmanGenRow][0]
= v0; wmaData[wmanGenRow][1] = v1; wmaData[wmanGenRow][2] = v2;
wmanGenRow++; pstepView->PostMessage(WM_WedDoStep2);}
void     WedMapAppend2(Locator&     llL,     JCellM*     pMap)
{if(wmaTime.Elapse(lagTime)
) {int maxPut = wmanGenRow; while(wmanPutRow != maxPut)
{if(!pMap->IsEmpty()) pMap->RowAppend(); int i = pMap->mRow2 -
2;
pMAPj0->SetRawData(wmaData[wmanPutRow][0]);
pMAPj1->SetRawData(wmaData[wmanPutRow][1]);
if(pMap->nCol2==3 && wmaData[wmanPutRow][2] != -1)
pMAPj2->SetRawData(wmaData[wmanPutRow][2]); pMap->fullDisplay =
FALSE;
JCellA* pwA = llL.pwA; pwA->AssureJCelldisplayable(pMAPj0);
WView* pView = pDoc->GetView(pwA->wType); if(pView)
pView->ScrollToIncludefullRect(pMap);   pwA->EBdef(pMap);   pwA-
>UpNoteMax();
pwA->UpPost(); wmanPutRow++;} wmaTime.Init();}}
void GetAverageCost(DSCS& dlg, prec profitSup0, prec profitSupX,
int  iProdSupply,  prec  quantX,  prec  priceX,  prec&  aCost)
{if(dlg.ac)
{aCost = profitSup0 - (profitSupX - quantX * priceX)
+ sumPayOt[iProdSupply] * quantX; if(!ZeroPress(aCost)) aCost =
0;
if(Divisible(aCost,   quantX)   &&   ZeroPress(quantX))   aCost/=
quantX;
else {if(!ZeroPress(quantX)) aCost = 0; else aCost = -1;}}
else aCost = -1;} int WedGenSupply(int step) {static DSCS dlg;
static JCellM* pMap; switch (step) {case 0: {dlg.llP.Init(llP);
if(dlg.DoModal()     ==     IDOK)     {pMap     =     (JCellM*)
llP.DGet(dlg.irowSupply,
colPSupply);    pMap->InitLoadReDo(!dlg.ac);    WedMapAppend0();
return IDOK;}
else {return IDCANCEL;}} case 1: {SetrwMax((long)
coln.DGet(rowFRWCaseIter)->VFetch(),  coln.DGet(rowFRWCaseTime)-
>VFetch());
```

```
int i, iCase; int iProdSupply = -1;
prec priceRef = llP.DGet(dlg.irowSupply, colPPrice)->VFetch();
prec profitSup0 = bigM; for(i=0;i<mProdw;i++) if(prod[i]
.wtProw0 == dlg.irowSupply) iProdSupply = i;
prec priceExeHold = priceExe[iProdSupply];
prec priceRawHold = priceRaw[iProdSupply]; if(dlg.ac) {WMInit();
WedDistInit(); priceExe[iProdSupply] = -1; priceRaw[iProdSupply]
= -1;
for(iCase=0;iCase<nSample;iCase++)          {pBase[iCase]->In();
CTLoadc(priceExe);
CTMaximize();           WedDistGen();          MaxPotential(bQt);
WMNoteScenario();}
profitSup0 = pProfitWM->GetProfit();} for(i=0;i<dlg.nPrice;i++)
{prec priceR = dlg.lo + i * (dlg.hi - dlg.lo)/ (dlg.nPrice - 1);
prec priceX = priceR + priceExeHold - priceRawHold; prec quant;
prec aCost = -1; if(IsEqual(priceR, priceRef)
|| dlg.lo <= priceRef && priceRef <= priceR)
{prec priceXref = priceRef + priceExeHold - priceRawHold;
quant = llP.DGet(dlg.irowSupply, colPMeanSupply)->VFetch();
GetAverageCost(dlg,    profitSup0,    profitBaseWM.GetProfit(),
iProdSupply,
quant, priceXref, aCost); WedMapAppend1(priceRef, quant, aCost);
if(IsEqual(priceR, priceRef)) {priceRef = -1; continue;}
else    priceRef    =    -1;}    if(0    <    ZeroPress(priceX))
{priceRaw[iProdSupply]
=   priceR;   priceExe[iProdSupply]   =   priceX;   WMInit();
WedDistInit();
for(iCase=0;iCase<nSample;iCase++)          {pBase[iCase]->In();
CTLoadc(priceExe);
CTMaximize();           WedDistGen();          MaxPotential(bQt);
WMNoteScenario();}
quant   =   cWM[iProdSupply].q.GetMean();   GetAverageCost(dlg,
profitSup0,
pProfitWM->GetProfit(), iProdSupply, quant, priceX, aCost);
WedMapAppend1(priceR,          quant,          aCost);}        else
{WedMapAppend1(priceR, 0, 0);}}
priceExe[iProdSupply] = priceExeHold; priceRaw[iProdSupply]
= priceRawHold; return 0;} case 2: {WedMapAppend2(llP, pMap);
return 0;}}
return -1;} void WedGenDemandPt(prec quant, int iPot, int jRC,
WedMeaner& wm) {prec qOriginal; if(iPot != -1) qOriginal =
bQt[iPot];
else qOriginal = rcQt[jRC]; WMInit(); WedDistInit(); for(int
iCase=0;
iCase<nSample;iCase++)     {pBase[iCase]->In();     WedDistGen();
if(iPot != -1)
{bQt[iPot] = quant; MaxPotential(bQt);} else {rcQt[jRC] = quant;
MaxPotentialqOrg(bQt, rcQt);} WMNoteScenario();} if(iPot != -1)
bQt[iPot]
= qOriginal; else rcQt[jRC] = qOriginal; wm = *pProfitWM; wm.ref
= quant;}
```

```
int WedGenDemand(int step) {static DDCS dlg; static JCellM*
pMap;
int cort = 1; int rcdt = 2; switch (step) {case 0: {int i;
ZEROOUT(dlg.row0Use[0],jcellRowsMax); for(i=0;i<mRS;i++)
dlg.row0Use[resSimple[i].wtRrow0] = cort; for(i=0;i<mRG;i++)
{llR.NoteIr(resGroup[i].wtRrow0); if(llR.p0(colRAvailability)
->IsString("Fixed")) dlg.row0Use[resGroup[i].wtRrow0] = rcdt;}
dlg.llR.Init(llR); if(dlg.DoModal() == IDOK) {pMap = (JCellM*)
llR.DGet(dlg.irowRes, colRDemand); pMap->InitLoadReDo(TRUE);
WedMapAppend0(); return IDOK;} else {return IDCANCEL;}}
case 1: {SetrwMax((long) coln.DGet(rowFRWCaseIter)->VFetch(),
coln.DGet(rowFRWCaseTime)->VFetch()); int i, j; prec offset;
if(1<dlg.nQuant) offset = ((dlg.hi - dlg.lo)/ (dlg.nQuant - 1))
* dlg.pc / 100.0; else offset = dlg.hi * dlg.pc / 100.0;
SPURTP(offset,
0.005); WedMeaner anti, post; int iPot = -1, jRC = -1;
int wolAdjustment = 0; if(dlg.row0Use[dlg.irowRes] == cort)
{FindAss(resSimple, wtRrow0, dlg.irowRes, i, mRS); iPot =
resSimple[i]
.cortRow; if(iPot == wolRow && wolAct) {iPot = -1; jRC = wolCol;
wolAdjustment = 1;} else if(resSimple[rsiCashRow]
.cortRow == iPot && !wolAct) {wolAdjustment = 1;}} else
{FindAss(resGroup,
wtRrow0, dlg.irowRes, j, mRG); jRC = resGroup[j].rcdtCol;}
for(i=0;
i<dlg.nQuant;i++) {prec quant; if(1<dlg.nQuant)
quant = dlg.lo + i * (dlg.hi - dlg.lo)/ (dlg.nQuant - 1);
else quant = dlg.lo; if(i==0 && !ZeroPress(quant))
{WedGenDemandPt(0,
iPot, jRC, anti); WedGenDemandPt(offset/4, iPot, jRC, post);}
else {prec qOff; qOff = quant - offset * 0.5; SPURTP(qOff, 0);
if(!IsEqual(qOff, post.ref)) WedGenDemandPt(qOff, iPot, jRC,
anti);
else {anti = post;} qOff = quant + offset * 0.5;
WedGenDemandPt(qOff,
iPot, jRC, post);} if(TRUE) {prec mv; prec prof =
post.v.GetMean()
- anti.v.GetMean(); prec qt = post.ref - anti.ref;
if(Divisible(prof, qt))
{mv = prof/qt; SPURTP(mv, 0); mv += wolAdjustment;
WedMapAppend1(mv,
quant, -1);}}} return 0;} case 2: {WedMapAppend2(llR, pMap);
return 0;}}
return -1;} void WedStep1(int (*pstepExecuteSet)(int), WView*
pView)
{pstepExecute = pstepExecuteSet; pstepView = pView; stepAct =
TRUE;
stepAbort = FALSE; if(TRUE) pstepThread =
AfxBeginThread(WedStep1Do,
&stepAbort); else WedStep1Do(&stepAct);} UINT WedStep1Do(LPVOID)
{try {lagTime = 5.0; (*pstepExecute)(1); lagTime = 0.1;
pstepView->SendMessage(WM_WedDoStep2);} catch(...)
```

```
{AfxMessageBox("Execution aborted.");} stepAct = FALSE;
pstepThread = NULL; pstepView->SendMessage(WM_WedDoStep3);
return 0;}
void WedStep2() {(*pstepExecute)(2);}
```

WHAT I CLAIM IS:

CLAIM 1. A system for allocating and optimizing usage/acquisition of organizational resources, comprising:

a computer having at least one processor, and memory means for storing data;

said memory means comprising a first memory portion storing a data base defining the available resources;

a second memory portion storing an iterative program for analyzing the data base based on selected criteria for optimizing resource allocation;

a third memory portion storing the quantitative resources available to the organization;

a fourth memory portion for storing a matrix array of organization groups utilizing the resources available to the organization;

a fifth memory portion for storing a resource-conduit program means for performing executable programs on said second, third, and fourth memory portions for optimizing the allocation of said organizational resources.

CLAIM 2. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 1, wherein said second memory portion further stores a linear programming process and at least updated values associated with said linear programming process.

CLAIM 3. The system for allocating and optimizing usage/acquisition of organizational resources according to claims 1 or 2, wherein said matrix array of organization groups utilizing the resources available to the organization comprises a matrix array of m rows and *nRes* columns; said *nRes* columns containing a plurality of resource-utilizing groups of the organization, said plurality of groups being arranged into at least one column of said matrix

array, said at least one column containing at least one resource-utilizing aspect associated with said respective group; said *m* rows of said matrix array including a plurality of said resource-utilizing aspects of at least two different said resource-utilizing aspect groups.

CLAIM 4. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 3, wherein said *m* rows of said matrix array is equal to at least some of the number of products of said organization plus at least some of the number of resources available.

CLAIM 5. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 4, wherein said fourth memory portion further stores a plurality of vector means for said matrix array, said plurality of vector means comprising a first vector means for storing the values of row effectiveness of each said *m* rows of said matrix array; a second vector means for storing the values of potential demand; a third vector means for storing the product of each element in said first and second vector means.

CLAIM 6. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 5, wherein said plurality of vector means for said matrix array further comprises additional vector means for operational and computational use by said resource-conduit program means of said fifth memory portion.

CLAIM 7. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 6, wherein said additional vector means comprises: *rwpDest*

vector, *rwpSour* vector, *rwOldAlloc* vector, and *rwOldMC* vector; each element in each said additional vector means applying only to a corresponding said column of said matrix array.

CLAIM 8. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 7, wherein said *rwpDest* vector is a RidgeWalk process destination operand containing pointers for shifting allocations to at least one said destination group; said *rwpSour* is a RidgeWalk process source operand containing pointers for shifting allocations from at least one said source group; said *rwOldAlloc* vector contains pre-allocation-shifting destination allocations; said *rwOldMC* vector contains source marginal costs.

CLAIM 9. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 7, wherein said fourth memory portion further stores a ***dpTie*** matrix array comprising one row for each of said plurality of products (*mProd*) and each said row containing indexes of said groups; and a *dpTieSubBlk* vector containing boolean values indicating whether said groups of said ***dpTie*** matrix array should not be used in said vector *rwpSour*.

CLAIM 10. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 3, wherein each of said plurality of resource-utilizing groups is arranged in said matrix array by column, at least one said group for a said column, each said group comprising at least a group head defining the data applicable to the entire said respective group; said group head containing all of the data fields of a group element; said group head also containing an allocation.

CLAIM 11. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 1 or 2, wherein resource-conduit program means of said fifth memory portion for performing executable programs on said second, third, and fourth memory portions comprises at least one of: first executable means for initially activating said iterative program of said second memory means for analyzing the data base based on initial criteria in order to initially allocate resources; second executable means for performing an AxisWalk for redistributing allocations among groups; third executable means for performing a TopWalk for redistributing allocations among groups; fourth executable means for performing a LateralWalk for redistributing allocations among groups; fifth executable means for performing a RidgeWalk for redistributing allocations among groups.

CLAIM 12. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 11, wherein said second executable program comprises redistributing resources from one group in a respective said column to another group in the same column of said matrix array.

CLAIM 13. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 11, wherein said third executable program comprises redistributing resources from at least one group to at least one group such that the mathematical product of the effectiveness of at least one pair of groups included in said redistribution remains constant.

CLAIM 14. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 11, wherein said fourth executable program comprises means for

temporarily changing potential demand data, triggering execution of second or third executable programs, and determining whether an improved allocation results.

CLAIM 15. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 11, wherein said fifth executable program comprises means for considering at least one of said products and transferring allocations between said groups of said matrix array in order to force an increase in the product's row-effectiveness.

CLAIM 16. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 11, wherein said first executable means for initially activating said iterative program of said second memory means for analyzing the data base based on initial criteria in order to initially allocate resources comprises means for loading said third memory means from said data base of said first memory means; means for loading said matrix array of organization groups utilizing the resources available to the organization; means for initially apportioning said third memory means between said groups of said matrix array; and means for generating measures of effectiveness of said groups and rows of said matrix array, whereby said resource-conduit program for optimizing resource allocations may be initiated.

CLAIM 17. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 1 or 2, wherein resource-conduit program means of said fifth memory portion for performing executable programs on said second, third, and fourth memory portions comprises: first executable means for performing an AxisWalk for redistributing allocations among groups; second executable means for performing a TopWalk for redistributing allocations among groups; third executable means for performing a RidgeWalk for redistributing allocations among groups.

CLAIM 18. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 1, 2, 3, or 10, wherein said fifth memory portion comprises a RandMode means for performing at least one of the following: generating initial instances; applying some or all of the walk processes to at least some instances; discarding instances with low absolute *d*; and randomly shifting the allocations between at least some groups within the same instance and evaluating whether such shifting increased absolute *d*; said RandMode means accepting as a final allocation the instance that yields the highest value of absolute *d*.

CLAIM 19. The system for allocating and optimizing usage/acquisition of organizational resources according to claim 18, wherein said fifth memory portion further comprises a GeneticMode means for combining allocations from different instances to form additional instances.

CLAIM 20. A means to allocate/acquire organizational resources comprising:

- A computer memory and CPU
- A linear programming process
- A Monte Carlo simulation process
- An application of obtained mean marginal values to price/cost/value resources.

CLAIM 21. The means according to claim 20 further comprising a resource-conduit process.

CLAIM 22. A means for an analyst to interactively allocate organizational resources consisting of:

- A computer memory and CPU
- A linear programming process
- A GUI that presents results in terms suggested by economic theory.

CLAIM 23. A method for allocating and optimizing usage acquisition of organizational resources, utilizing a computer having at least one processor, and memory means for storing data. said method comprising:

(a) storing a data base defining the available resources in a first portion of the memory means;

(b) storing in a second portion of the memory means an iterative program for analyzing the data base based on selected criteria for optimizing resource allocation;

(c) storing in a third portion of the memory means the quantitative resources available to the organization;

(d) storing in a fourth portion of the memory means a matrix array of organization groups utilizing the resources available to the organization; and

(e) storing in a fifth memory portion a resource-conduit program means for performing executable programs on said second, third, and fourth memory portions for iteratively optimizing the allocation of said organizational resources.

CLAIM 24. The method for allocating and optimizing usage of organizational resources according to claim 23, wherein said step (b) comprises storing a linear programming process, and further comprising storing in a sixth portion of the memory means at least the updated values associated with the linear programming process of said step (b) as determined by said step (e).

CLAIM 25. The method for allocating and optimizing usage of organizational resources according to claims 23 or 24, wherein said step (d) for storing a matrix array of organization groups utilizing the resources available to the organization comprises a matrix array of m rows and n columns; said n columns containing a plurality of resource-utilizing groups of the organization, said plurality of groups being arranged into at least one column of said matrix array, said at least one column containing at least one resource utilizing aspect associated with said respective group; said n RES rows of said matrix array including a plurality of said resource utilizing aspects of at least two different said resource-utilizing aspect groups.

CLAIM 26. The method for allocating and optimizing usage of organizational resources according to claim 25, wherein said said m rows of said matrix array is equal to at least some of the number of products of said organization plus at least some of the number of resources available.

CLAIM 27. The method for allocating and optimizing usage of organizational resources according to claim 26, wherein said step (d) further stores a plurality of vector means for said matrix array, said plurality of vector means comprising a first vector means for storing the temporary value for use in said iterative program of said second memory portion; a second vector means for storing the current said iterative program's original vector value; a third vector means for storing the values of row-effectiveness of each said row of said m rows of said matrix array; a fourth vector means for the values of potential demand; said current vector value of said second vector means being the product of each element in said third and fourth vector means.

CLAIM 28. The method for allocating and optimizing usage of organizational resources according to claim 27, wherein said plurality of vector means for said matrix array further comprises additional vector means for operational and computational use by said resource-conduit program means of said fifth memory portion.

CLAIM 29. The method for allocating and optimizing usage of organizational resources according to claim 28, wherein said additional vector means comprises: rwpDest vector, rwpSour vector, rwOldAlloc vector, rwOldMC vector, and dpTieSubBlk vector; each element in each said additional vector means applying only to a corresponding said column of said matrix array.

CLAIM 30. The method for allocating and optimizing usage of organizational resources according to claim 29, wherein said rwpDest vector is a ridge-walk process destination operand containing pointers for shifting allocations to at least one said destination group; said rwpSour is a ridge-walk process source operator containing pointers for shifting allocations from at least one said source group; said rwOldAlloc vector contains pre-allocation-shifting destination allocations; said rwOldMC vector contains source marginal costs.

CLAIM 31. The method for allocating and optimizing usage of organizational resources according to claim 29, wherein said step (d) further stores a dpTie Matrix array comprising one row for each of said plurality of products (mProd) containing indexes of groups said additional vector dpTieSubBlk containing boolean values indicating whether said groups having only a single element of said dpTie matrix array should not be used in said vector rwpSour.

CLAIM 32. The method for allocating and optimizing usage of organizational resources according to claim 25, wherein each of said plurality of resource-utilizing groups is arranged in said matrix array by column, at least one said group for a said column, each said group comprising a group head defining the data applicable to the entire said respective group, and at least one group element; said group head containing all of the data fields of a group element; said group head also containing an allocation and a variable to hold working-temporary allocation values.

CLAIM 33. The method for allocating and optimizing usage of organizational resources according to claim 23 or 24, wherein resource-conduit program means of said fifth memory portion for performing executable programs on said second, third, and fourth memory portions comprises at least one of: first executable means for initially activating said iterative program of said second memory means for analyzing the data base based on initial criteria in order to initially maximally optimize resources; second executable means for performing an axis walk for redistributing allocations among groups; third executable means for performing a top walk for redistributing allocations among groups; fourth executable means for performing a lateral walk for redistributing allocations among groups; fifth executable means for performing a RidgeWalk for redistributing allocations among groups.

CLAIM 34. The method for allocating and optimizing usage of organizational resources according to claim 33, wherein said second executable program comprises redistributing resources from one group in a respective said column to another group in the same column of said matrix array.

CLAIM 35. The method for allocating and optimizing usage of organizational resources according to claim 33, wherein said third executable program comprises redistributing resources from at least one group in a respective said column to another group in the same column of said matrix array such that the mathematical product of any particular two groups' effectiveness remains constant.

CLAIM 36. The method for allocating and optimizing usage of organizational resources according to claim 33, wherein said fourth executable program comprises means for evaluating the results of said second and third executable programs to check for inter-dependency between said second and third executable programs which may result in an instantaneous desirable quantum change in one of said second and third executable programs upon starting a shift or movement of allocation.

CLAIM 37. The method for allocating and optimizing usage of organizational resources according to claim 33, wherein said fifth executable program comprises means for considering at least one of said products and transferring allocations to said groups of said matrix array in order to force an increase in the product's row-effectiveness.

CLAIM 38. The method for allocating and optimizing usage of organizational resources according to claim 33, wherein said first executable means for initially activating said iterative program of said second memory means for analyzing the data base based on initial criteria in order to initially optimize resources comprises means for loading said third memory means from said data base of said first memory means; means for loading said matrix array of organization groups utilizing the resources available to the organization of said fourth memory portion; means for initially apportioning said third memory means between said groups of said matrix array; and means for generating measures of effectiveness of said groups and rows of

said matrix array; means for loading values for said iterative program of said second memory portion, whereby said iterative program for analyzing the data base based on selected criteria for optimizing resource allocation may be initially executed using the initial values for determining an initial maximization of resource allocation.

CLAIM 39. The system for allocating and optimizing usage of organizational resources according to claim 23 or 24, wherein resource-conduit program means of said fifth memory portion for performing executable programs on said second, third, and fourth memory portions comprises : first executable means for performing an axis walk for redistributing allocations among groups; second executable means for performing a top walk for redistributing allocations among groups; third executable means for performing a ridge walk for redistributing allocations among groups.

CLAIM 40. The method for allocating and optimizing usage of organizational resources according to claim 23, 24, 25 or 32, wherein said fifth memory portion comprises RandWalk means for performing at least one of the following: applying some or all of the Walk processes to at least some instances; creating additional instances; disguarding instances with low absolute d; and randomly perturbing the allocations of at least some groups and evaluating whether such perturbances increase absolute d; said RandWalk means accepting as a final allocation the instance that yields the highest value of absolute d.

CLAIM 41. The method for allocating and optimizing usage of organizational resources according to claim 40, wherein said fifth memory portion further comprises VarWalk means for combining allocations from different instances to form additional instances.

FIG. 1

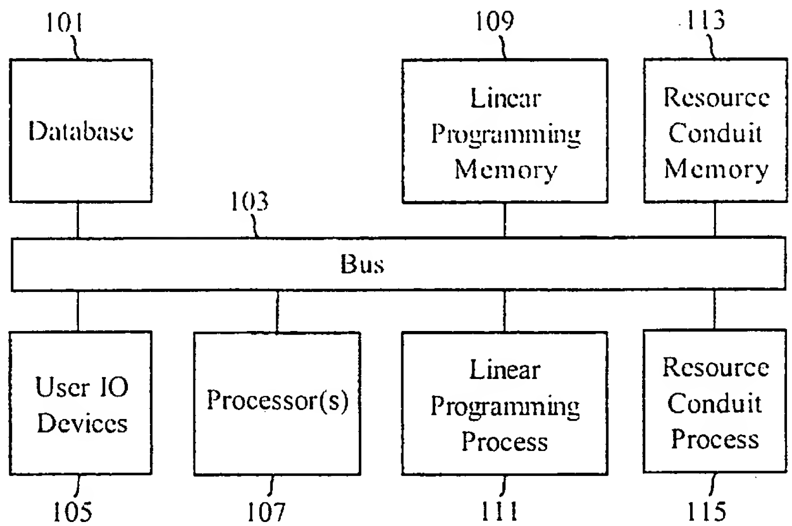

101
Database
109
Linear
Programming
Memory
113
Resource
Conduit
Memory
103
Bus
User IO
Devices
Processor(s)
Linear
Programming
Process
Resource
Conduit
Process
105
107
111
115

FIG. 2

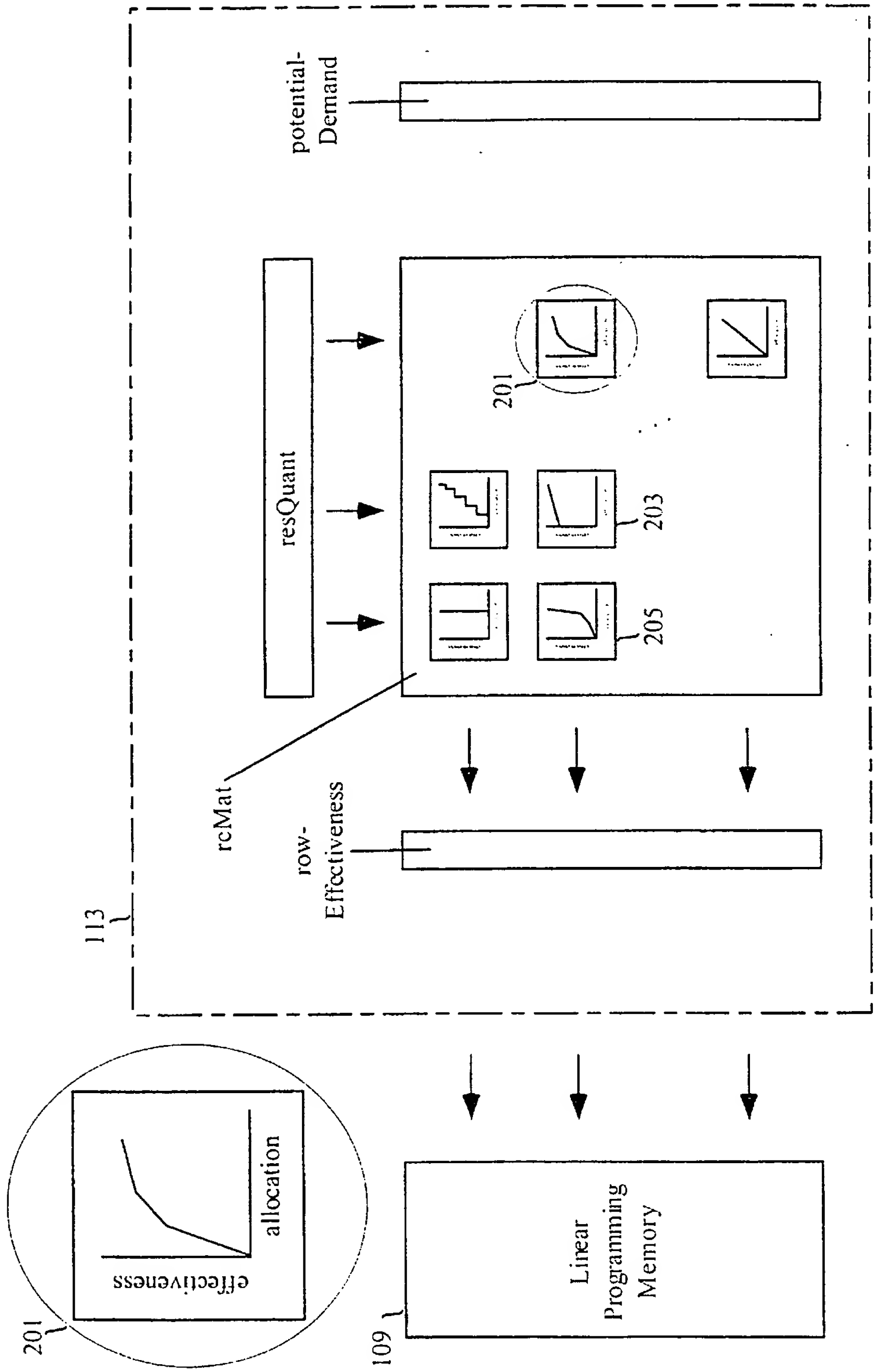

potential-
Demand
resQuant
rcMat
row-
Effectiveness
113
201
203
205
effectiveness
allocation
201
Linear
Programming
Memory
109

FIG. 3

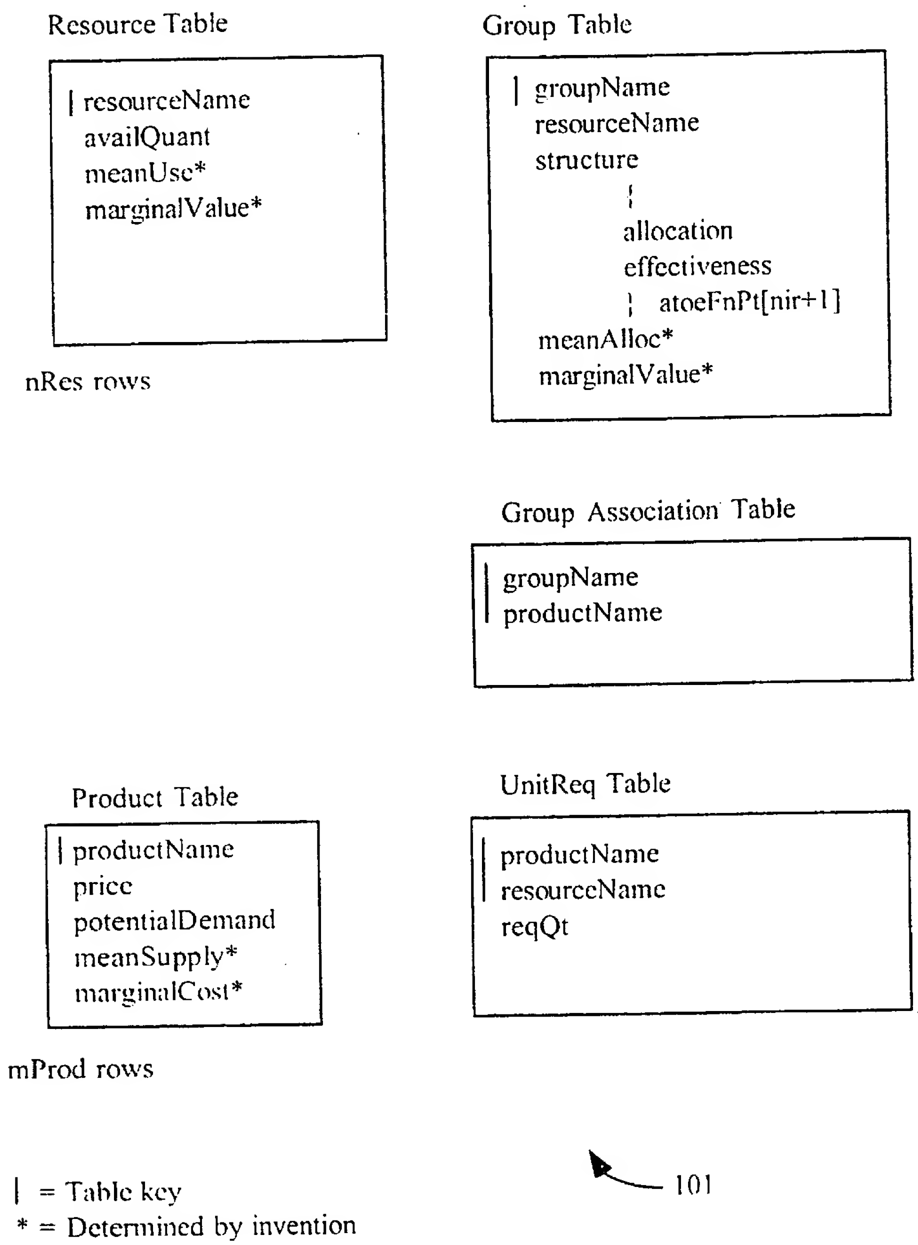

Resource Table
| resourceName
availQuant
meanUse*
marginalValue*
nRes rows
Group Table
| groupName
resourceName
structure
allocation
effectiveness
atoeFnPt[nir+1]
meanAlloc*
marginalValue*
Group Association Table
groupName
productName
Product Table
| productName
price
potentialDemand
meanSupply*
marginalCost*
mProd rows
UnitReq Table
productName
resourceName
reqQt
101
| = Table key
* = Determined by invention

FIG. 4
(Prior Art)

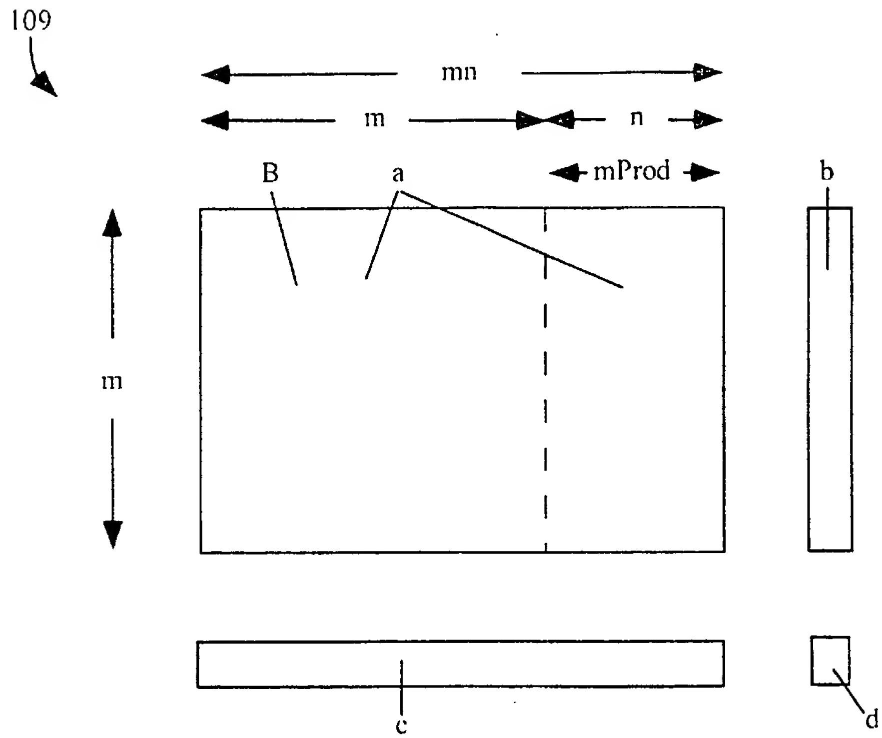


FIG. 5

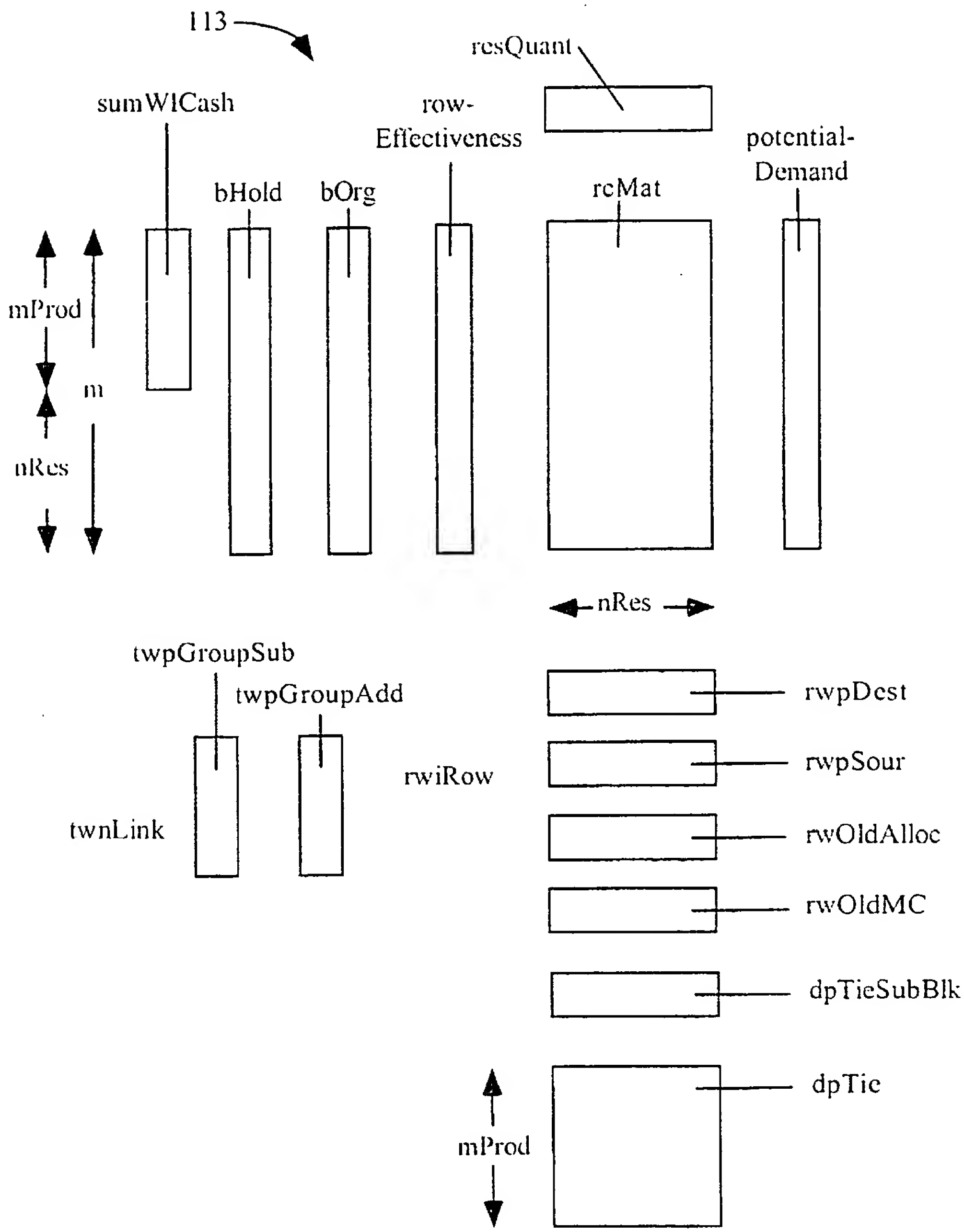

113
resQuant
sumWICash
row-
Effectiveness
potential-
Demand
bHold
bOrg
rcMat
mProd
m
nRes
nRes
twpGroupSub
twpGroupAdd
rwiRow
twnLink
rwpDest
rwpSour
rwOldAlloc
rwOldMC
dpTieSubBlk
dpTie
mProd

FIG. 6

Group Head

```
allocation
allocationHold
structure
        {
        allocation
        effectiveness
        }  atocFnPt[nir+1]
ir
maxSub
maxAdd
dedaSub
dedaAdd
gmcSub
gmvAdd
twmcSub
twcRow
twcCol
twcsRow
effectivenessHold

subBlk
effectiveness
emcSub
emvAdd
```

Group Element

```
subBlk
effectiveness
emcSub
emvAdd
```

FIG. 7

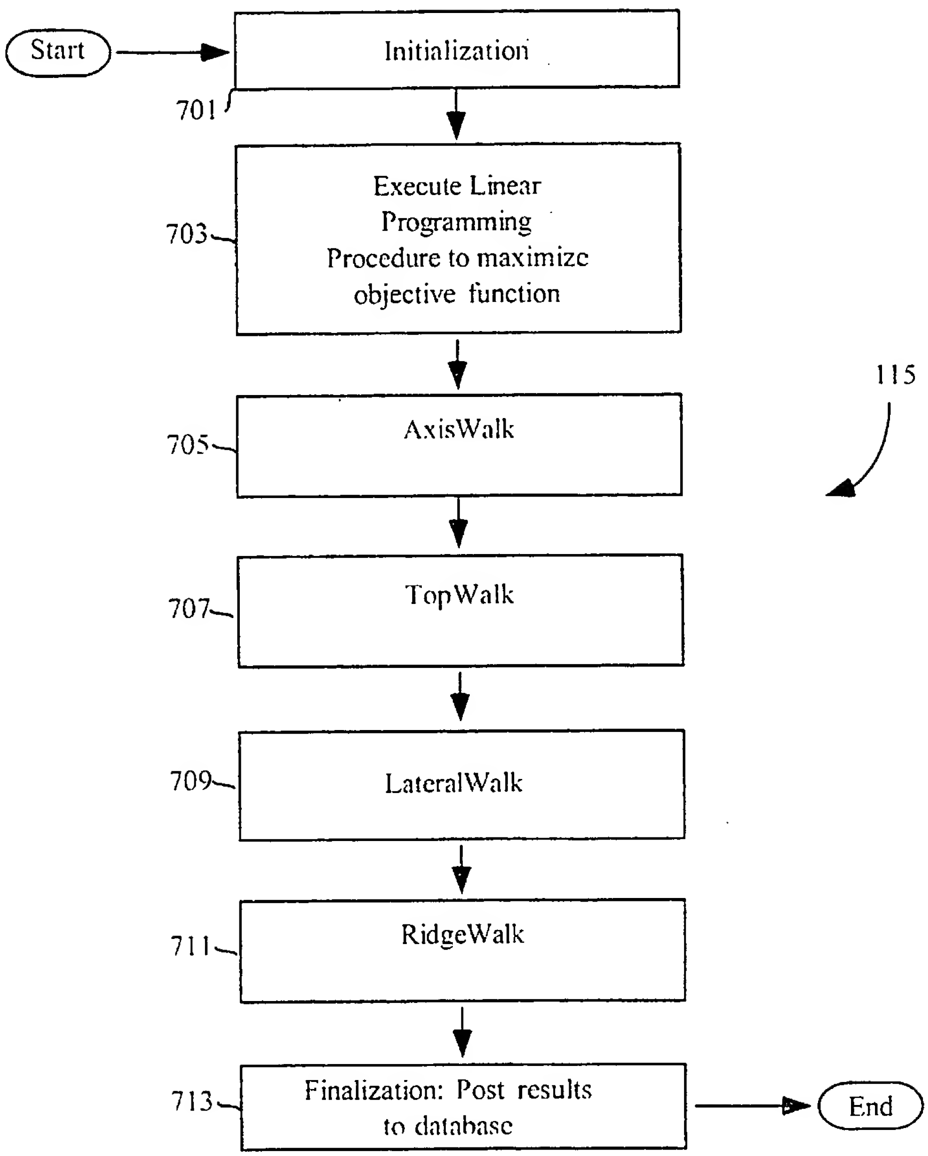

Start
Initialization
701
Execute Linear Programming Procedure to maximize objective function
703
AxisWalk
705
TopWalk
707
LateralWalk
709
RidgeWalk
711
Finalization: Post results to database
713
End
115

FIG. 8A

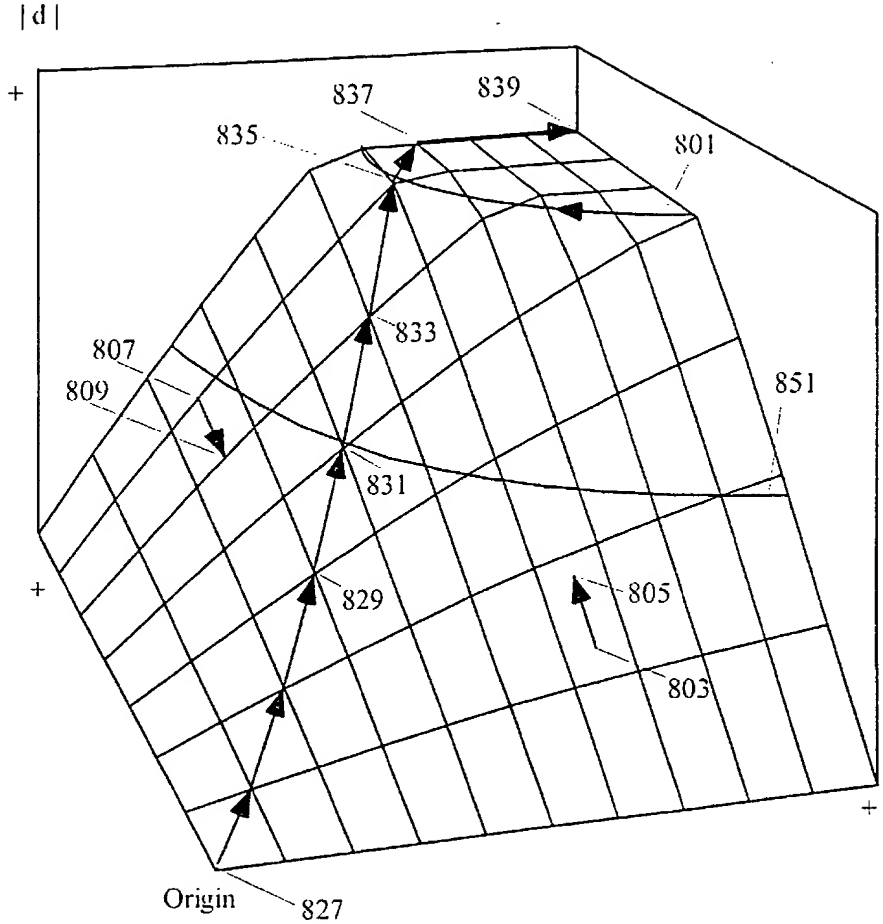

|d|
+
837
839
835
801
833
807
809
851
831
+
829
805
803
+
Origin
827

FIG. 8B

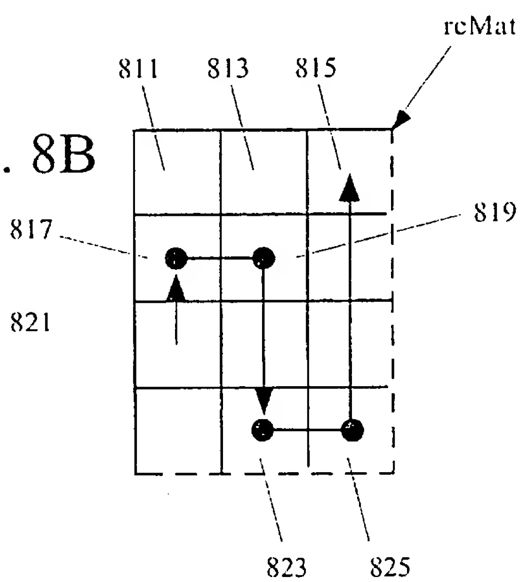

rcMat
811
813
815
817
819
821
823
825

FIG. 9

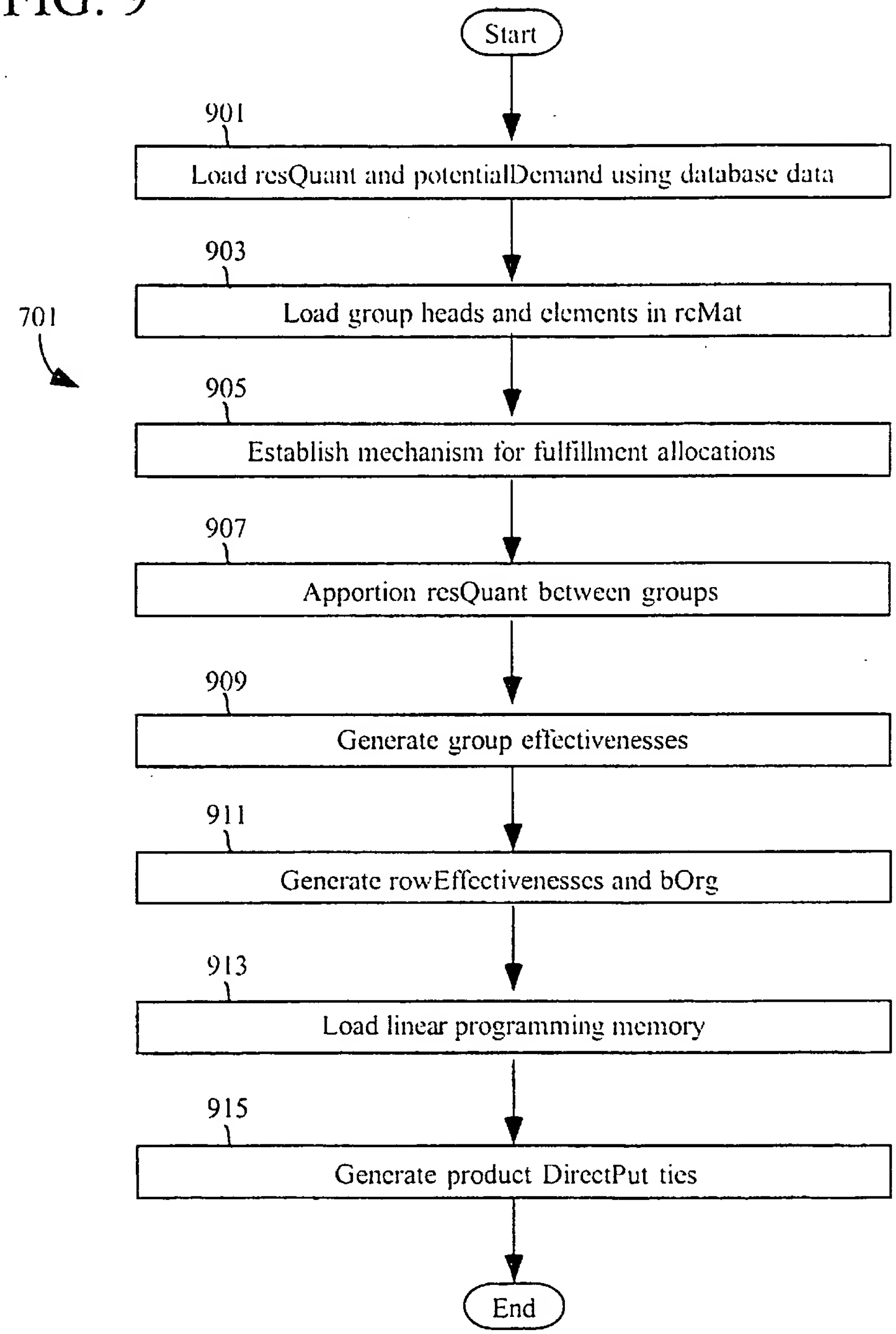

Start
901
Load resQuant and potentialDemand using database data
903
Load group heads and elements in reMat
701
905
Establish mechanism for fulfillment allocations
907
Apportion resQuant between groups
909
Generate group effectivenesses
911
Generate rowEffectivenesses and bOrg
913
Load linear programming memory
915
Generate product DirectPut ties
End

FIG. 10

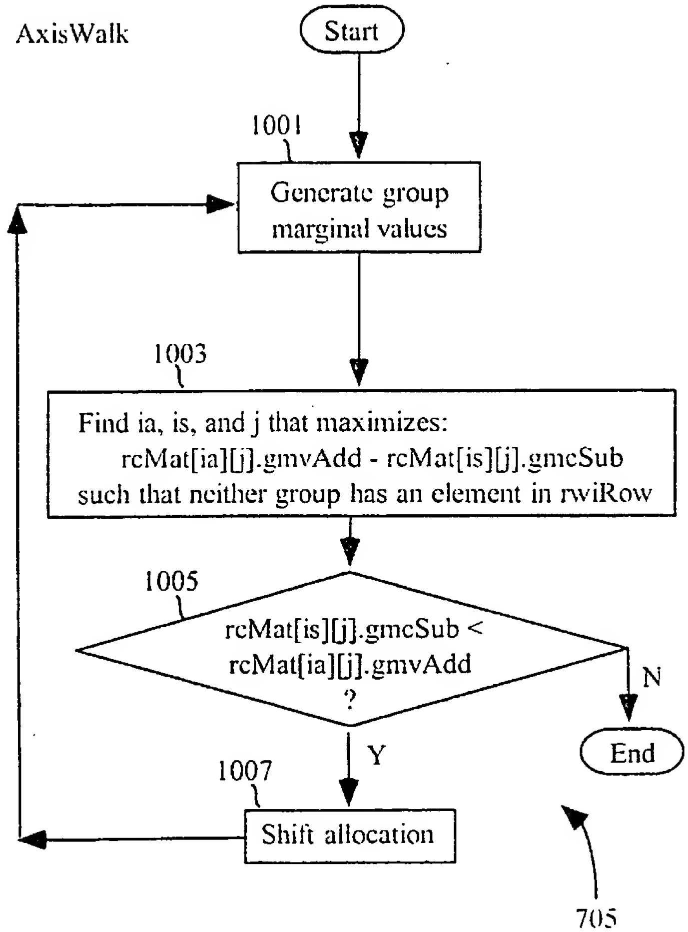

AxisWalk
Start
1001
Generate group marginal values
1003
Find ia, is, and j that maximizes:
rcMat[ia][j].gmvAdd - rcMat[is][j].gmcSub
such that neither group has an element in rwiRow
1005
rcMat[is][j].gmcSub <
rcMat[ia][j].gmvAdd
?
N
End
Y
1007
Shift allocation
705

FIG. 11A

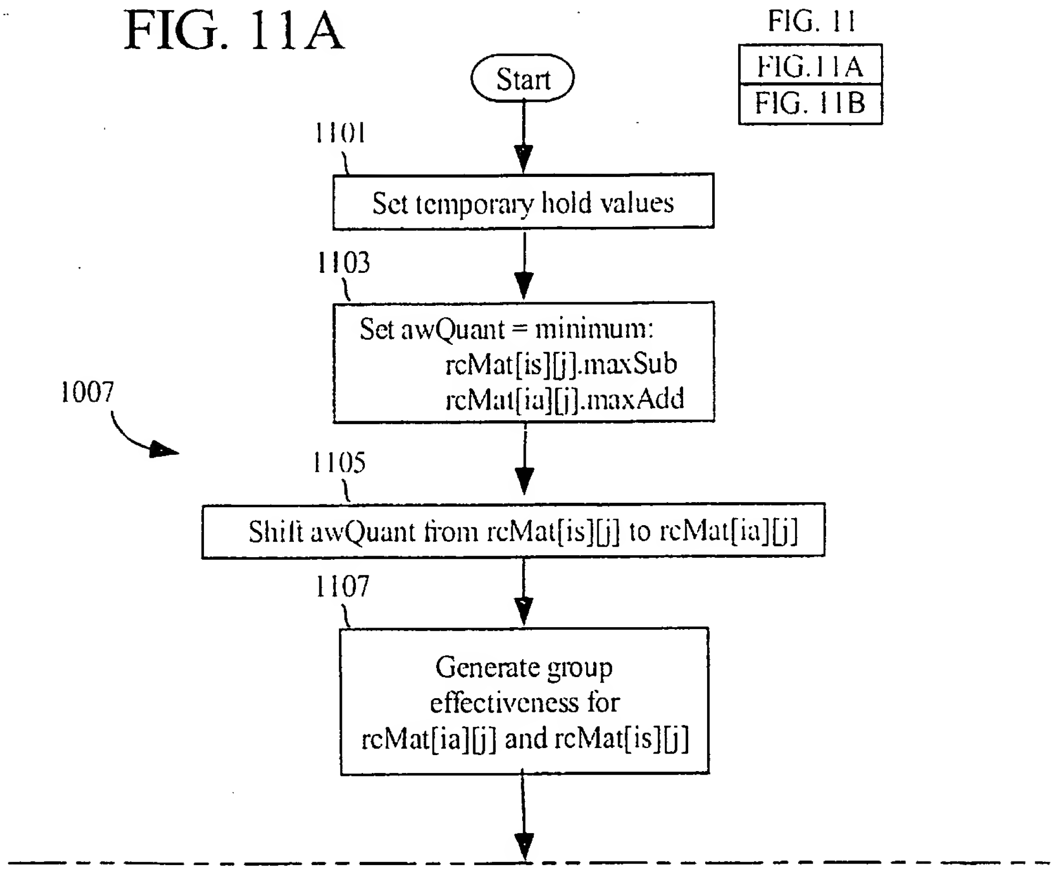


FIG. 11
FIG.11A
FIG. 11B
Start
1101
Set temporary hold values
1103
Set awQuant = minimum:
rcMat[is][j].maxSub
rcMat[ia][j].maxAdd
1007
1105
Shift awQuant from rcMat[is][j] to rcMat[ia][j]
1107
Generate group
effectiveness for
rcMat[ia][j] and rcMat[is][j]

FIG. 11B

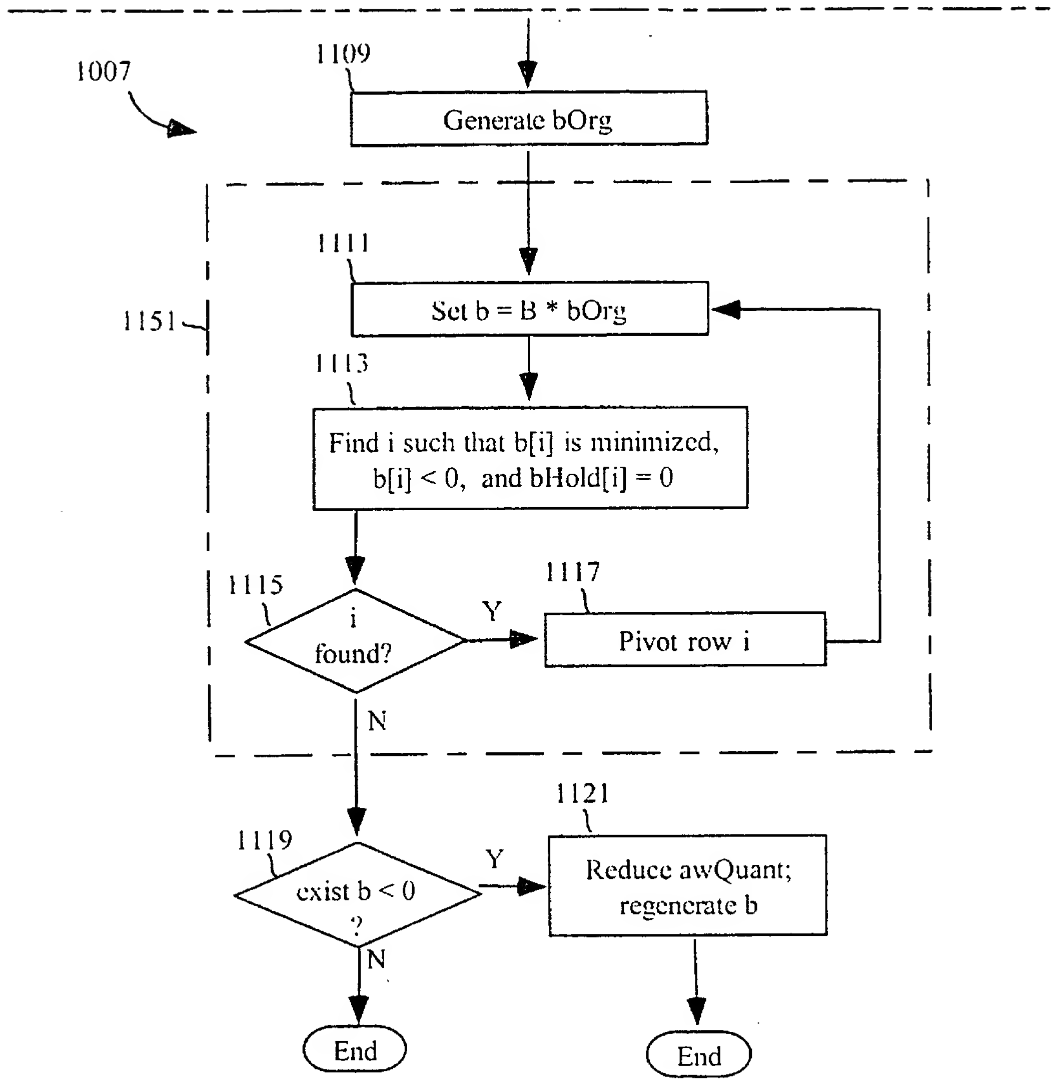

1007
1109
Generate bOrg
1151
1111
Set b = B * bOrg
1113
Find i such that b[i] is minimized,
b[i] < 0, and bHold[i] = 0
1115
i
found?
Y
1117
Pivot row i
N
1121
1119
exist b < 0
?
Y
Reduce awQuant;
regenerate b
N
End
End

FIG. 12

TopWalk

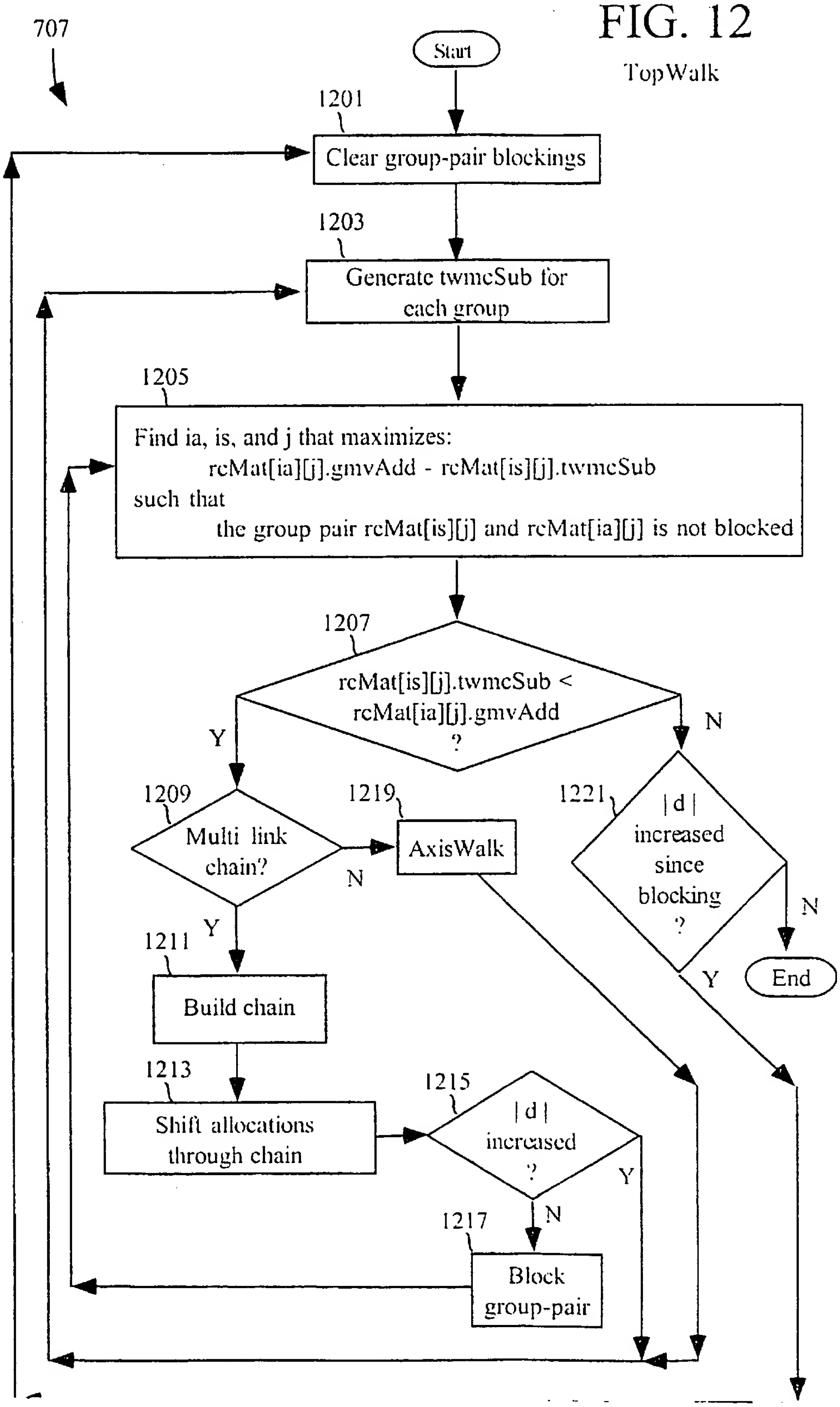
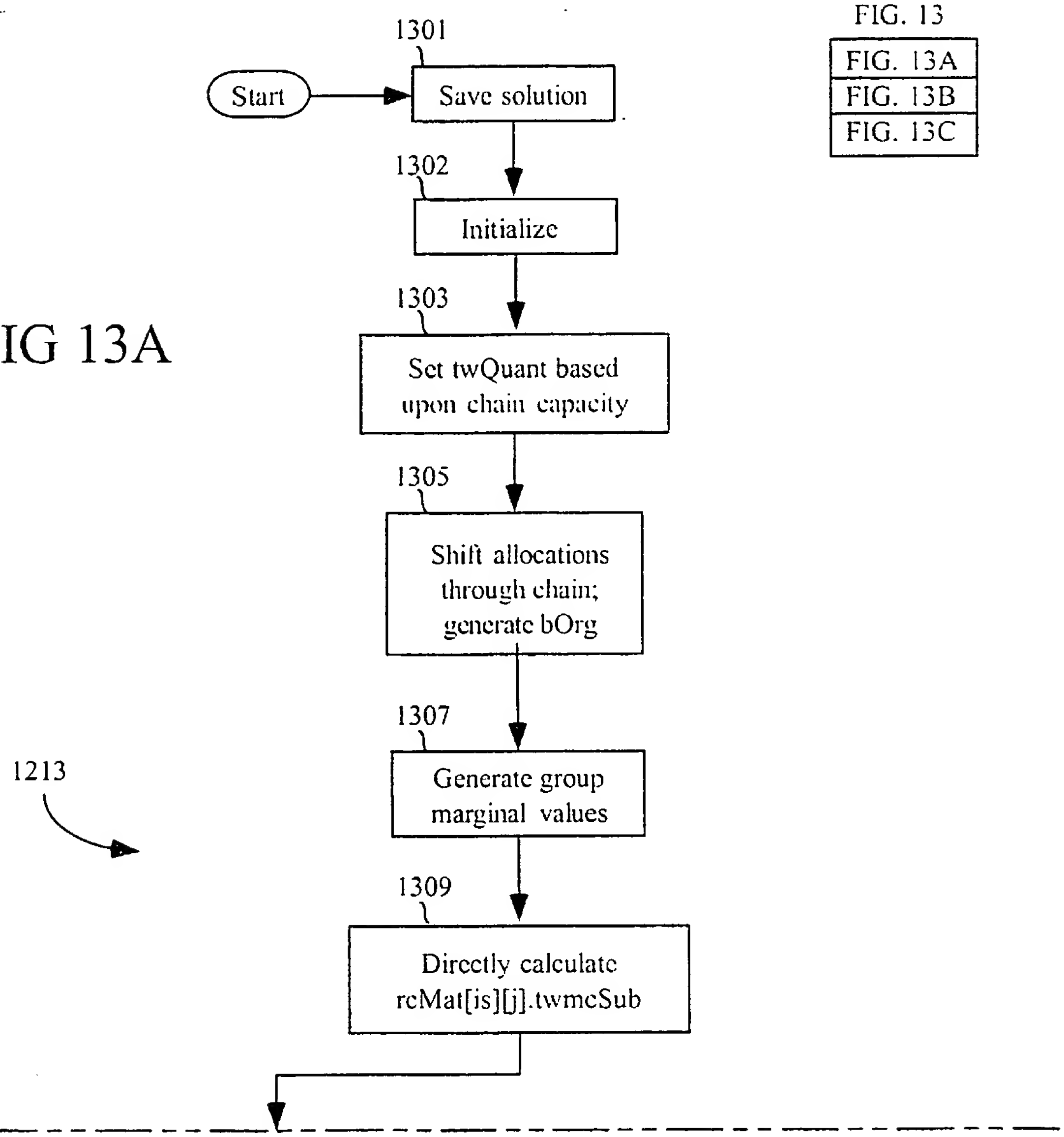


FIG. 13

FIG. 13A
FIG. 13B
FIG. 13C

FIG 13A



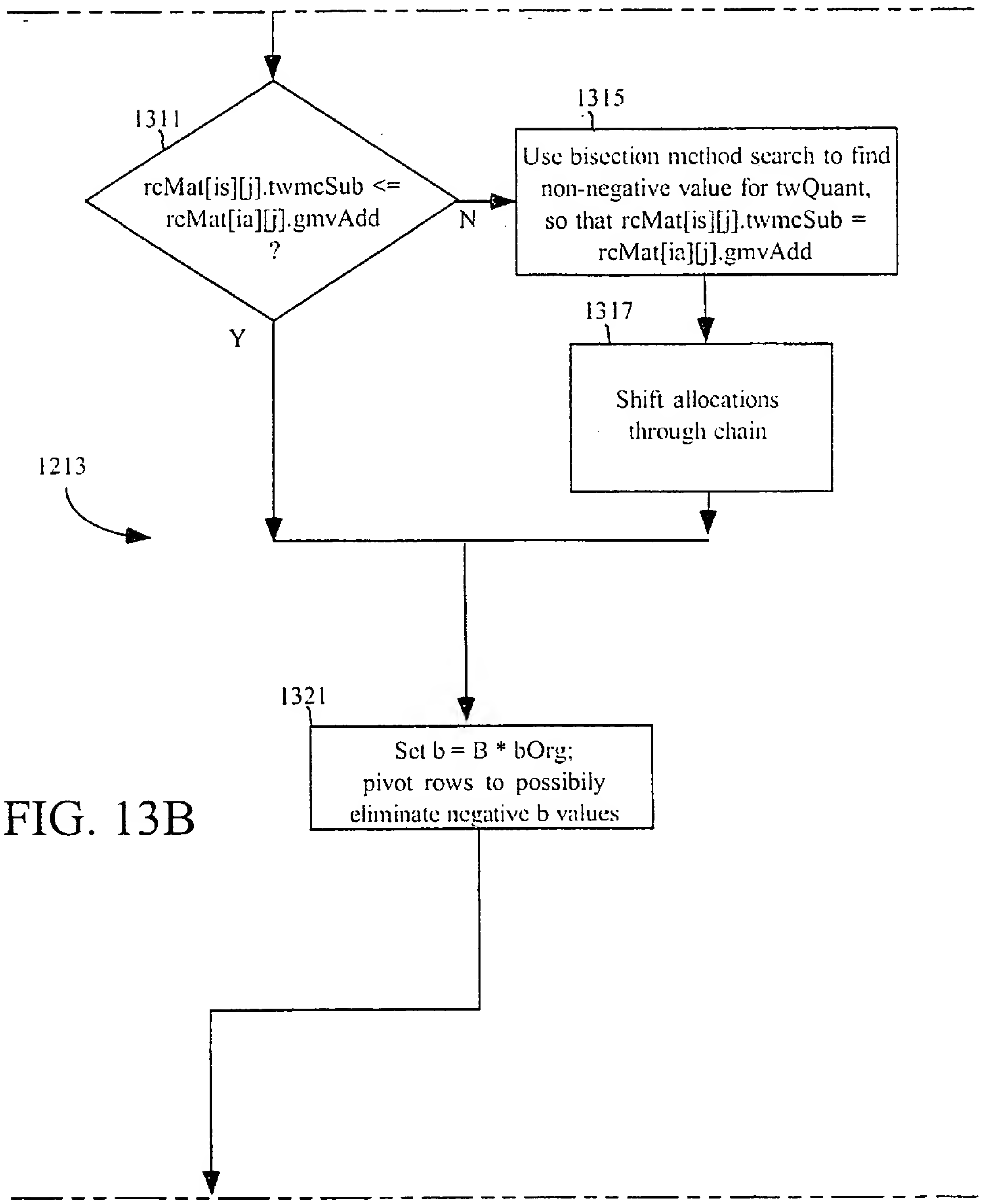

1311
rcMat[is][j].twmcSub <=
rcMat[ia][j].gmvAdd
?
N
Y
1315
Use bisection method search to find non-negative value for twQuant, so that rcMat[is][j].twmcSub = rcMat[ia][j].gmvAdd
1317
Shift allocations through chain
1213
1321
Set b = B * bOrg;
pivot rows to possibily eliminate negative b values

FIG. 13B

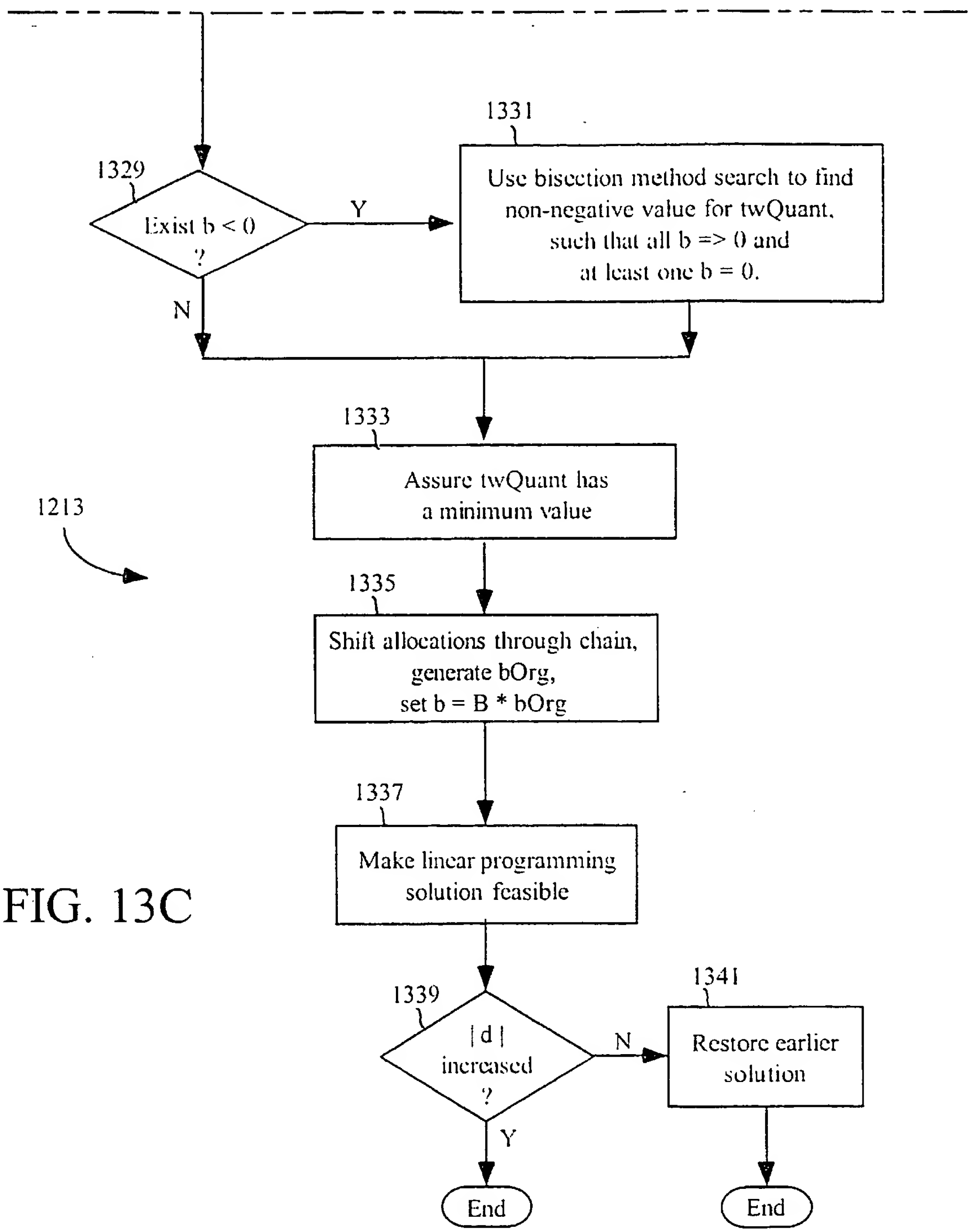

1331
Use bisection method search to find non-negative value for twQuant, such that all b => 0 and at least one b = 0.
1329
Exist b < 0 ?
Y
N
1333
Assure twQuant has a minimum value
1213
1335
Shift allocations through chain, generate bOrg, set b = B * bOrg
1337
Make linear programming solution feasible
1339
| d | increased ?
N
Y
1341
Restore earlier solution
End
End

FIG. 13C

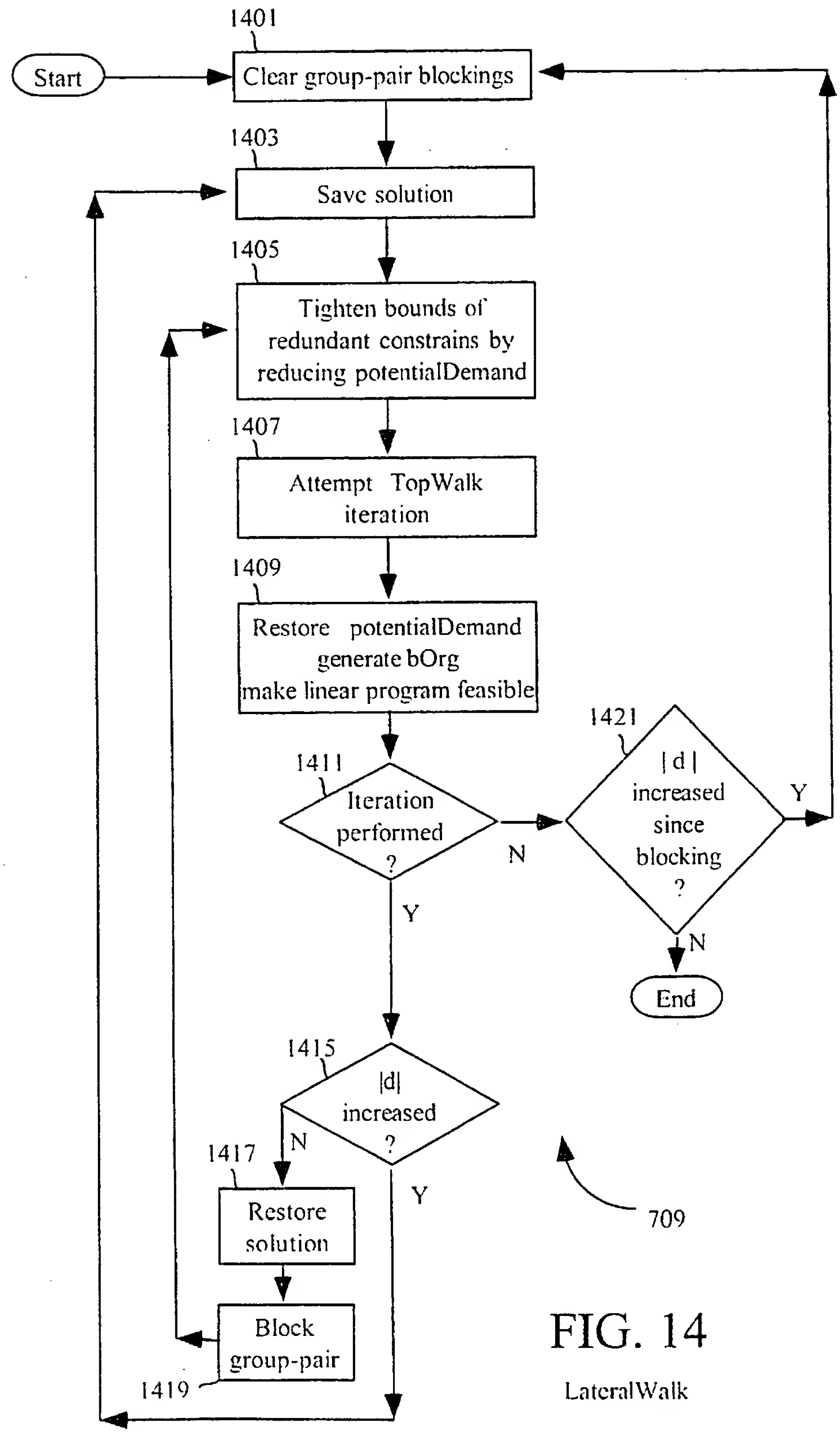


FIG. 14

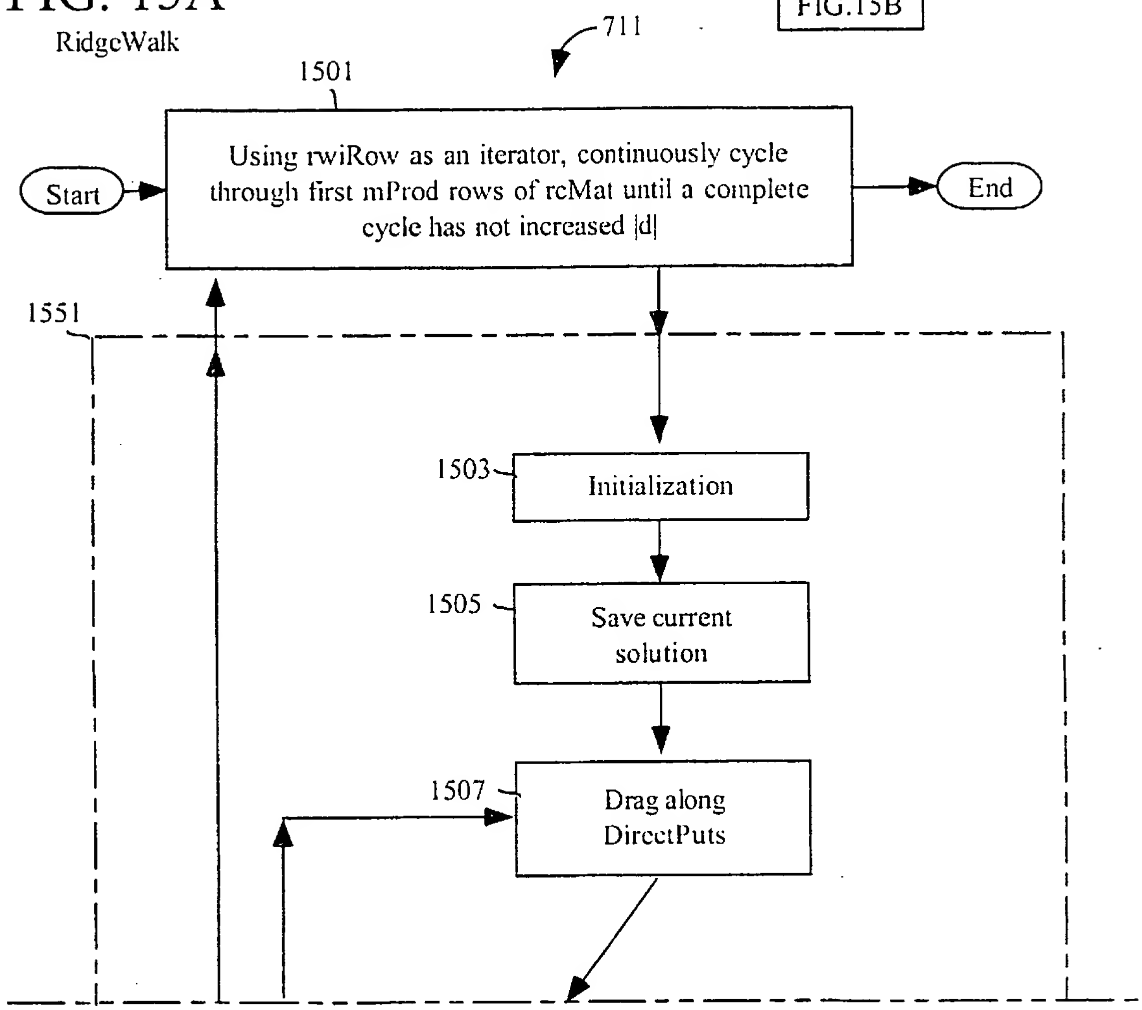
LateralWalk

FIG. 15

FIG.15A
FIG.15B

FIG. 15A

RidgeWalk



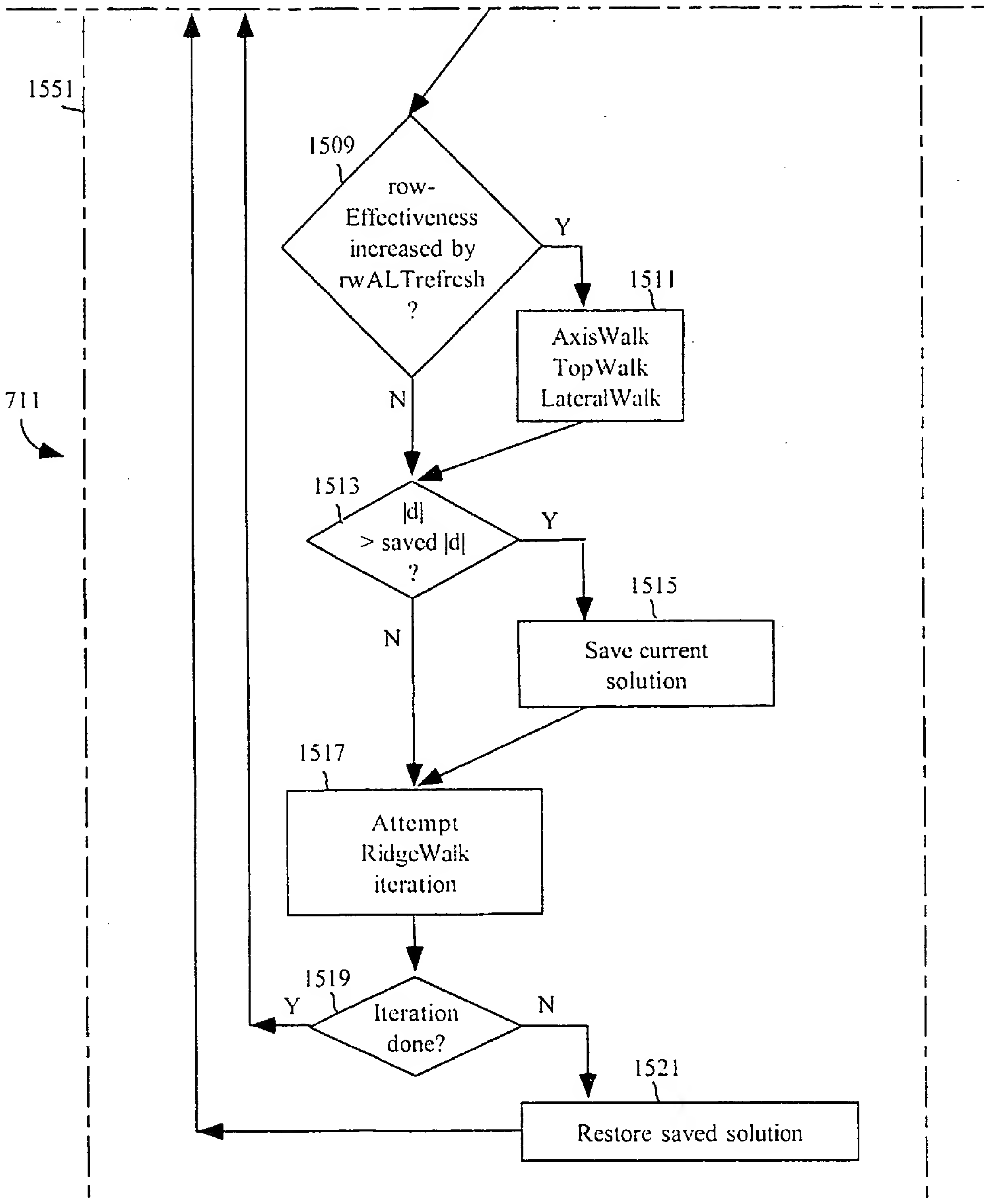
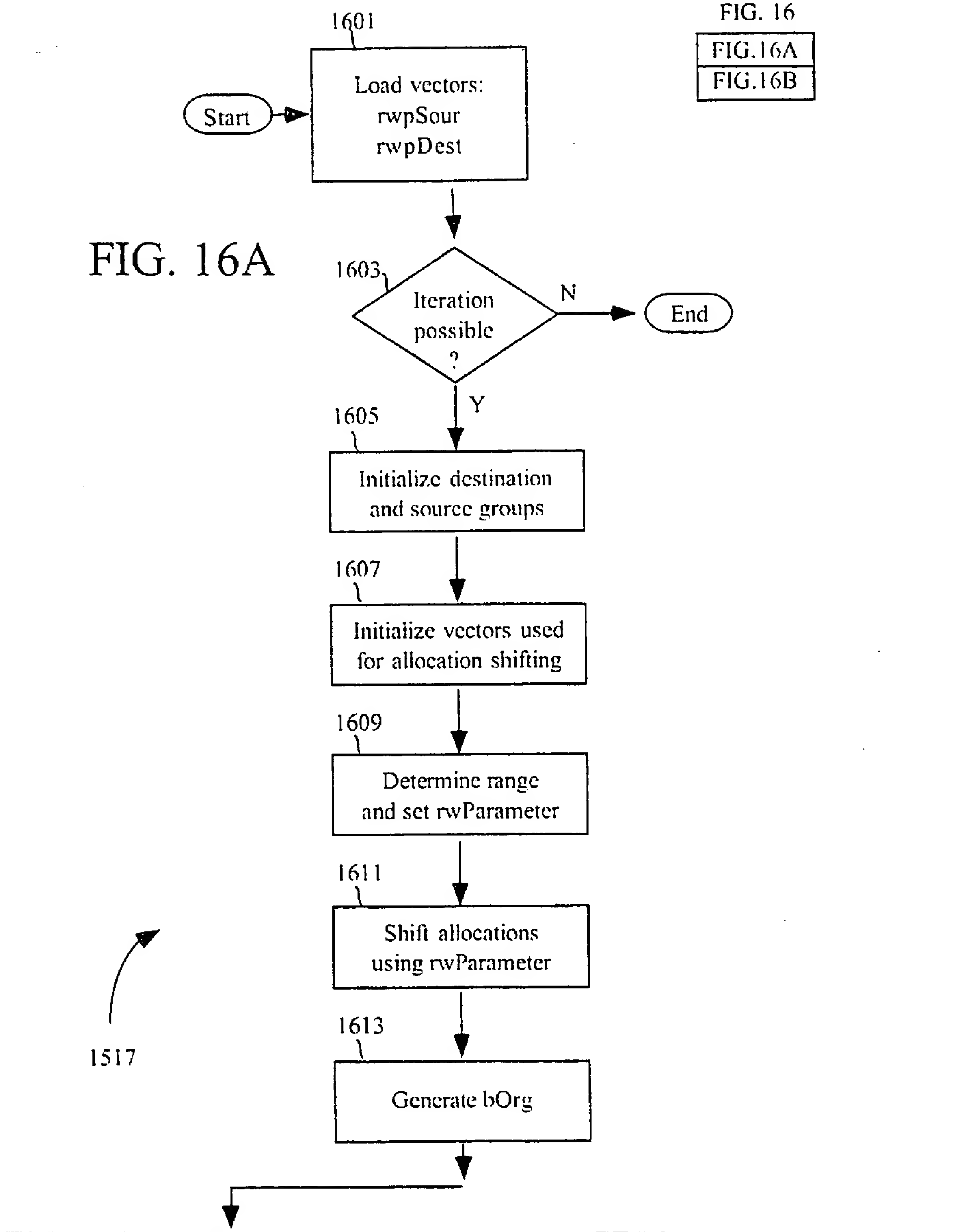

1551
711
1509
row-Effectiveness increased by rwALTrefresh ?
Y
N
1511
AxisWalk
TopWalk
LateralWalk
1513
|d| > saved |d| ?
Y
N
1515
Save current solution
1517
Attempt RidgeWalk iteration
1519
Iteration done?
Y
N
1521
Restore saved solution

FIG. 15B

FIG. 16

FIG.16A
FIG.16B

FIG. 16A



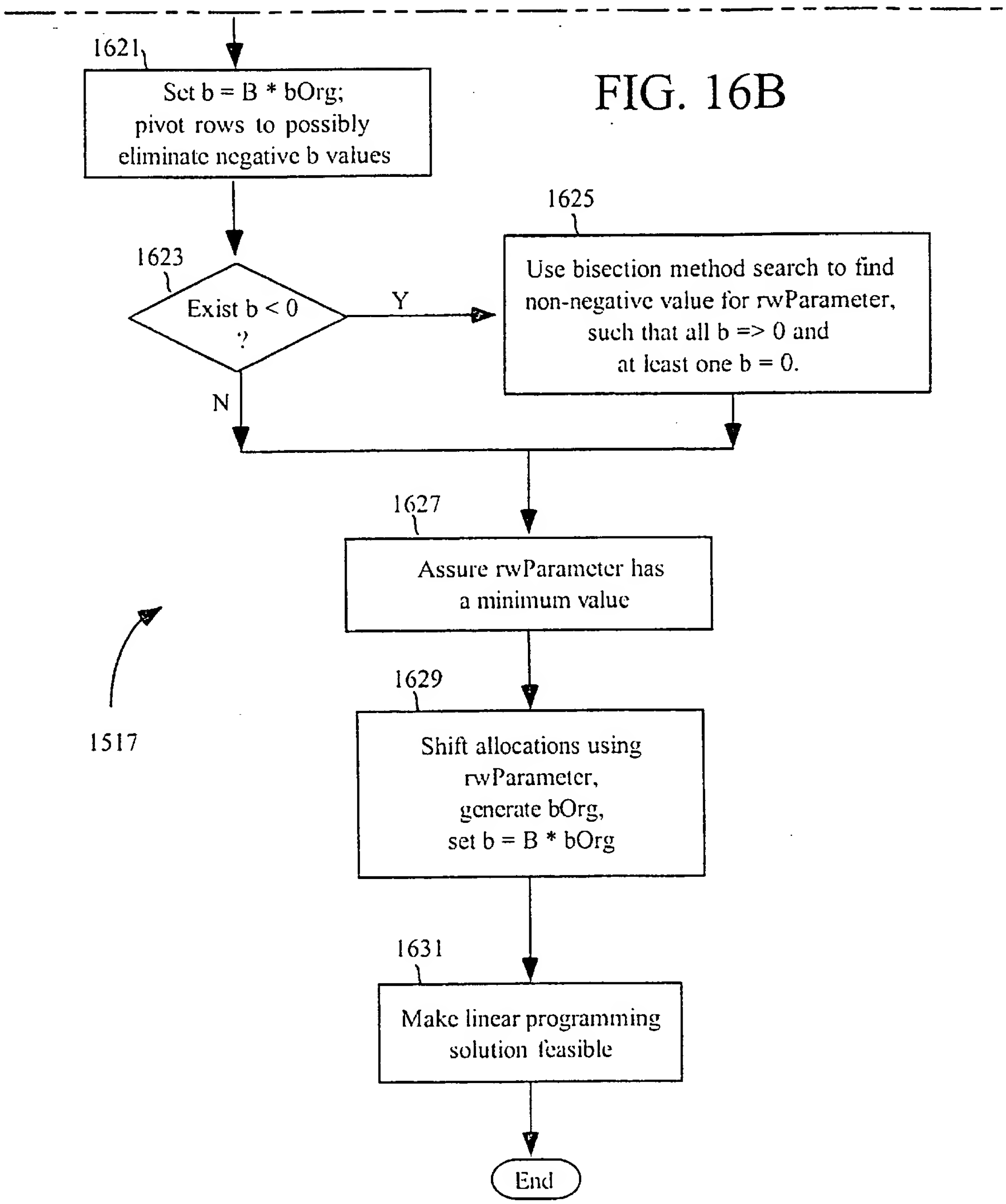

1621
Set b = B * bOrg;
pivot rows to possibly
eliminate negative b values
FIG. 16B
1623
Exist b < 0
?
Y
N
1625
Use bisection method search to find
non-negative value for rwParameter,
such that all b => 0 and
at least one b = 0.
1627
Assure rwParameter has
a minimum value
1517
1629
Shift allocations using
rwParameter,
generate bOrg,
set b = B * bOrg
1631
Make linear programming
solution feasible
End

FIG. 17

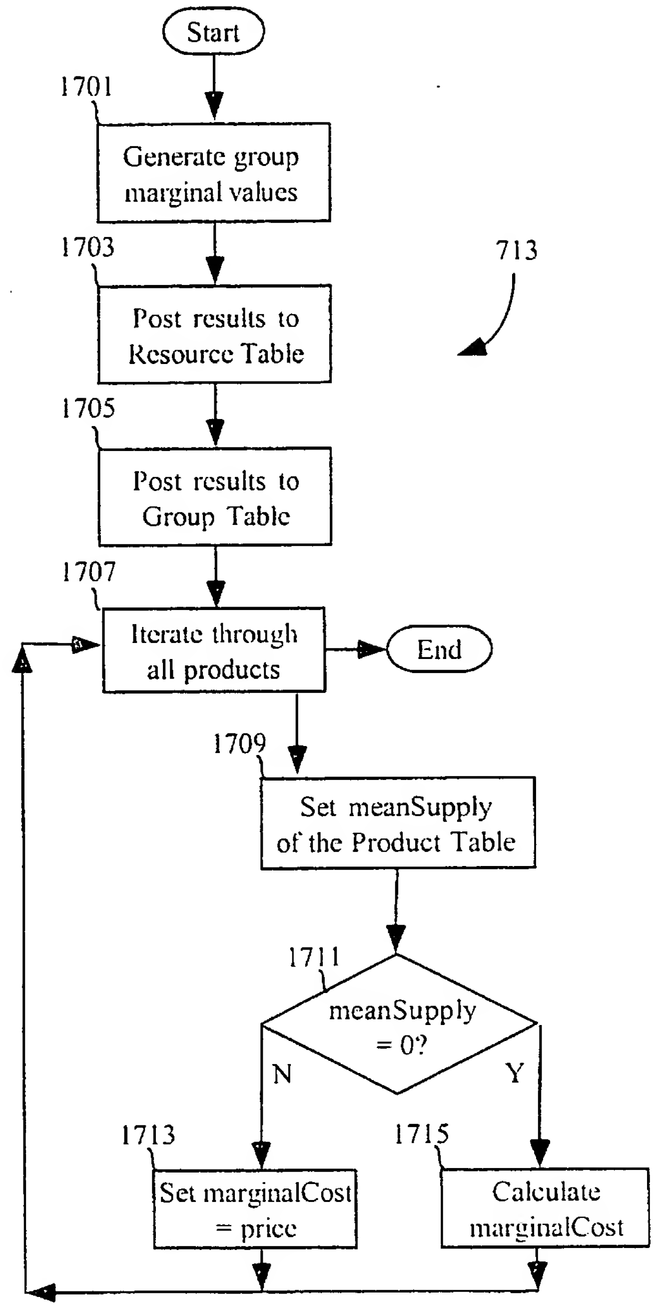

Start
1701
Generate group marginal values
1703
Post results to Resource Table
713
1705
Post results to Group Table
1707
Iterate through all products
End
1709
Set meanSupply of the Product Table
1711
meanSupply = 0?
N
Y
1713
Set marginalCost = price
1715
Calculate marginalCost

FIG. 18

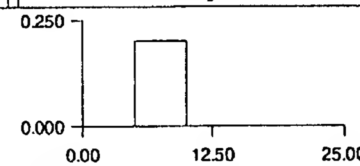
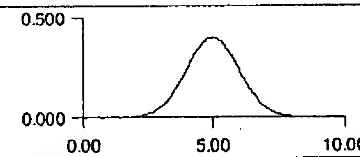
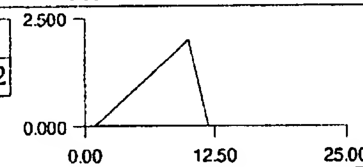
distName	distType				distObject	meanGen	marginalValue
Uniform	Uniform	Low 5	High 10			8	$0
Normal100	Normal	Mean 5	S.D. 1			5	$0
Triangle100	Triangular	Low 1	Mode 10	High 12		8	$850

FIG. 19

FIG. 19A	FIG. 19B

FIG. 19A

resourceName	unit	availability	availQuant	WTMD	payPrice	
groupName	fixedAlloc fxAlQt				atoeObject	
WI Cash		Fixed	$1000			
Building Space	1000 sq ft day	Fixed	10			
Designer	days	Fixed	20			
Service Design	No					
Overnight Air	package	Buyable			$20	

Quantity	Effectiveness(%)
0	0%
15	20%
20	100%

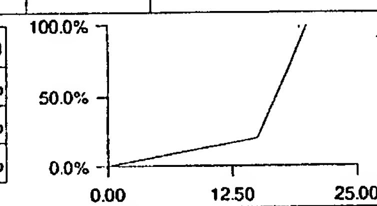


FIG. 19B

marginalValue	meanUse	demandObject
marginalValue	meanAlloc	
$0	600	
$1540	10	
$1043	20	
$1043	20	
	30	

Price	Quantity
1540.00	10.00
1540.00	12.00
1540.00	14.00
128.33	16.00
0.00	18.00
0.00	20.00

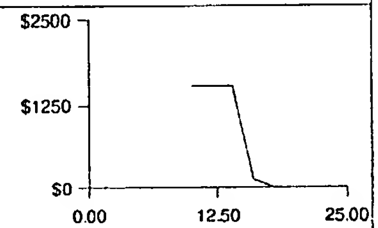


FIG. 20

FIG. 20A	FIG. 20B

FIG. 20A

productName / resourceName	price / reqQt	fillValue / periodsToCash	distName	dist%	carryOver
Widget	$5000	$0	Normal100	100%	20%
Manager	0.2				
Building Space	4.0				
Associate/Labor/Staff	1.0				
Rail	1.0				
Widget Design					
Equipment	5.0				
Service-- Non Profit	$50	$1600	Uniform	100%	90%
Manager	0.4				
Building Space	1.0				
Associate/Labor/Staff	5.0				
Service Design					

FIG. 20B

marginalCost	meanPotDemand	meanSupply	supplyObject
$9658	5.0	0.0	

Price	Quantity	AC
0.00	0.00	0.00
5000.00	0.00	0.00
10000.00	2.50	8886.67
15000.00	2.50	8886.67
20000.00	2.50	8886.67
25000.00	2.50	8886.67

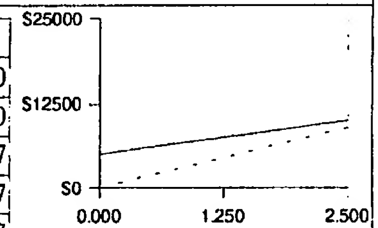
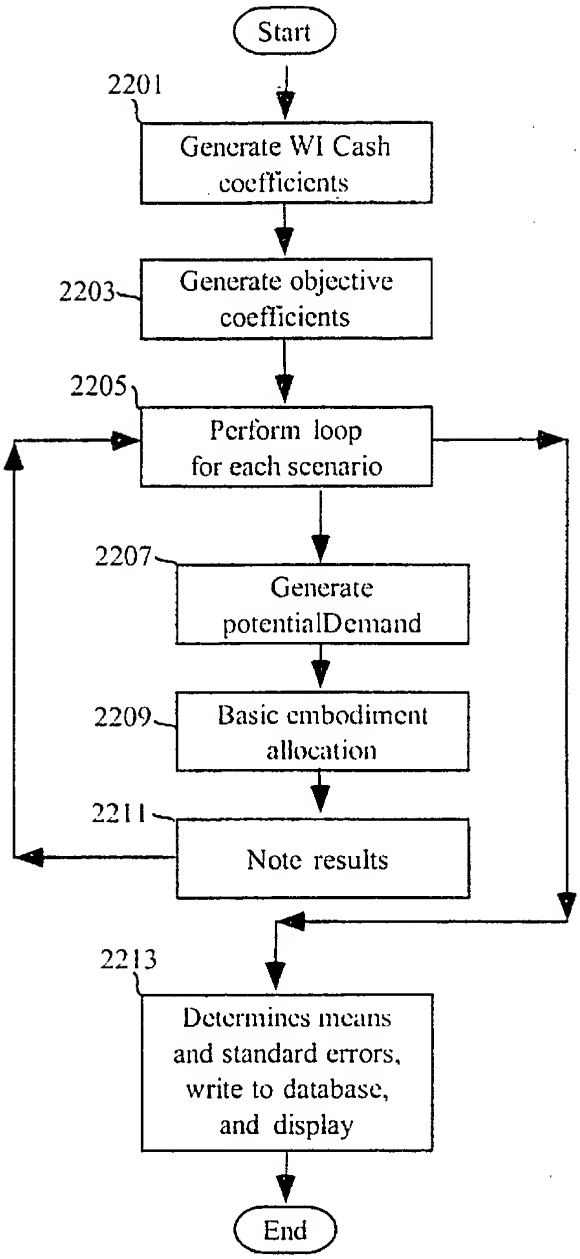


FIG. 21

	Next	Previous (0)	Current
Internal Producer's Surplus		$21719.34	$21916.67
Standard Error		173.35	-
Change in WI Cash		$18114.56	$18683.33
Standard Error		399.27	-
WI Cash (Beginning)		$1000.00	$1000.00
Marginal Value		$0.00	$0.00
Standard Error		0.00	-
Sum Fill Value		$4104.78	$3733.33
Standard Error		254.75	-
Sum WTMD		$500.00	$500.00
Standard Error		0.00	-
Parameters			
Allocation	Direct	Indirect	Direct
Maximization	IPS	IPS	IPS
WI Cash Type	Spread Out	Spread Out	Spread Out
Rand Seed		1	
N Sample		200	
MC/MV Display	Partial	Partial	Partial
Base			
Max RW Iterations	3	3	3
Max RW Time (sec)	20.000	20.000	20.000
Supply/Demand			
Max RW Iterations	3	3	3
Max RW Time (sec)	20.000	20.000	20.000

FIG. 22


Start
2201
Generate WI Cash coefficients
2203
Generate objective coefficients
2205
Perform loop for each scenario
2207
Generate potentialDemand
2209
Basic embodiment allocation
2211
Note results
2213
Determines means and standard errors, write to database, and display
End

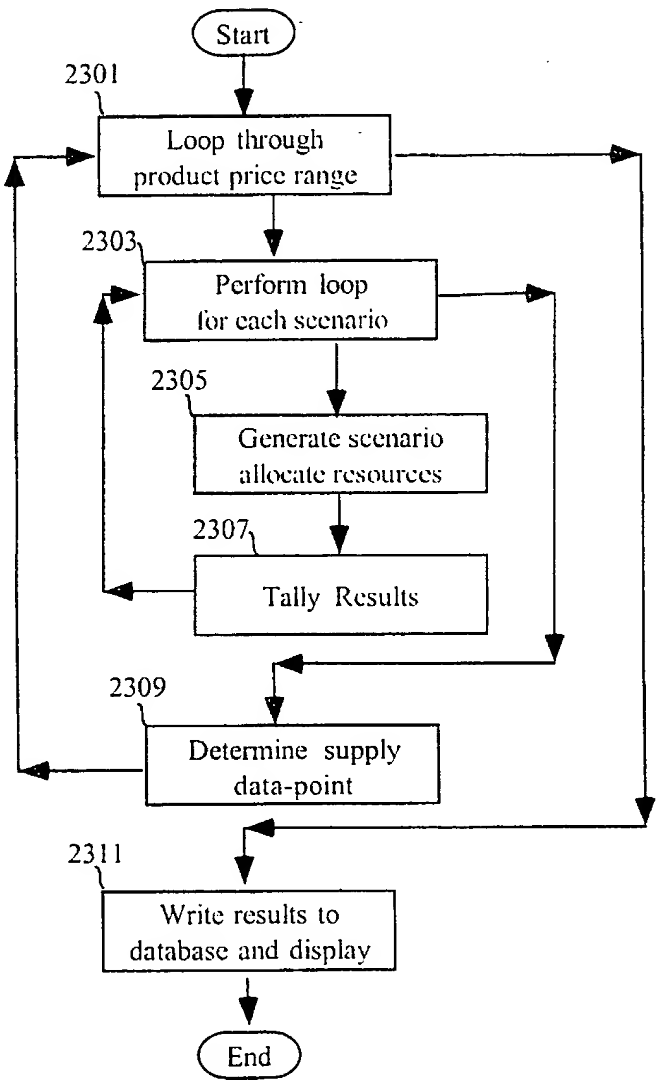
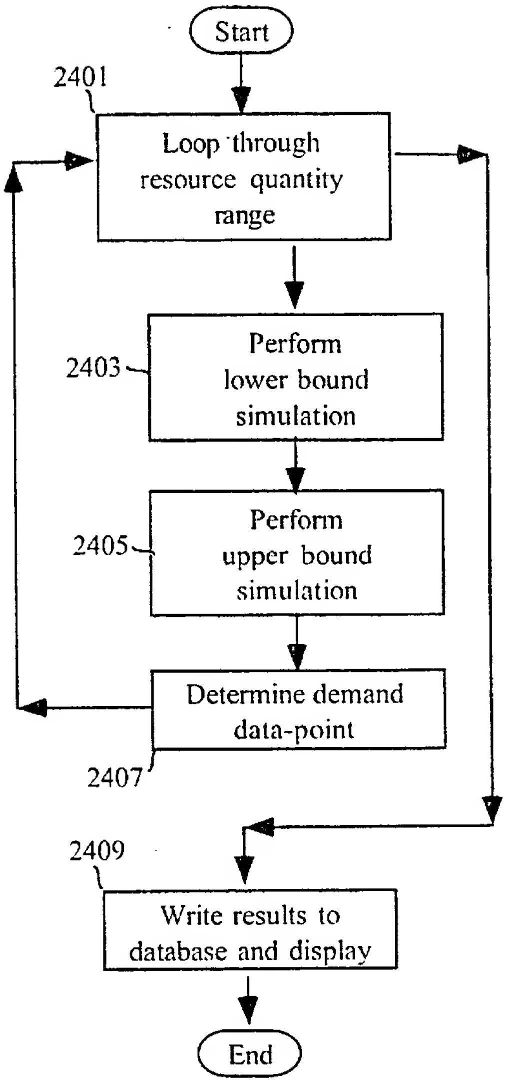

Start
2301
Loop through product price range
2303
Perform loop for each scenario
2305
Generate scenario allocate resources
2307
Tally Results
2309
Determine supply data-point
2311
Write results to database and display
End

FIG. 23

FIG. 24


Start
2401
Loop through resource quantity range
2403
Perform lower bound simulation
2405
Perform upper bound simulation
Determine demand data-point
2407
2409
Write results to database and display
End

INTERNATIONAL SEARCH REPORT

International Application No
P.T/US 98/09009

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/60

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 615 109 A (EDER JEFF) 25 March 1997 see abstract; claim 1; figure 2 see column 27, line 48 - column 29, line 9 see column 32, line 60 - column 33, line 2; figure 4 see column 67, line 39 - column 68, line 38 see column 94, line 8 - line 54 ---	1,2,23,24
A	EP 0 306 965 A (INTELLIMED CORP) 15 March 1989 see abstract see column 8, line 15 - line 37 --- -/--	1,23

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
17 September 1998	24/09/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Suendermann, R

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

In' ıtional Application No
PCT/US 98/09009

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 672 990 A (MINNESOTA MINING & MFG) 20 September 1995 see abstract see page 5, line 30 - line 37; figure 2 ---	22
X	US 5 255 345 A (SHAEFER CRAIG G) 19 October 1993 see abstract; claims 1,9 -----	20,21

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 98/09009

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 5615109	A	25-03-1997	NONE		
EP 0306965	A	15-03-1989	US	4937743 A	26-06-1990
			AU	2206088 A	23-03-1989
			CA	1294054 A	07-01-1992
			JP	1162977 A	27-06-1989
			JP	2634070 B	23-07-1997
EP 0672990	A	20-09-1995	US	5623404 A	22-04-1997
			CA	2142501 A	19-09-1995
			IL	112552 A	04-01-1998
			JP	8030694 A	02-02-1996
US 5255345	A	19-10-1993	US	5222192 A	22-06-1993